Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 157

Infrastructure support for augmented memory

Paul Metzger

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Albrecht Schmidt
Supervisor:	Prof. Dr. Nigel Davies Dr. Alireza Sahami Dipl.-Medieninf. Tilman Dingler
Commenced:	May 27, 2014
Completed:	November 25, 2014
CR-Classification:	H.3.4

Acknowledgments

I would like to thank Prof. Dr. Albrecht Schmidt and Tilman Dingler for giving me the opportunity to work on my Bachelor thesis in the context of the European project RECALL at the Lancaster University.

Thank you to Prof. Dr. Nigel Davies for his support, patience and supervision at the Lancaster University. He gave me valuable feedback throughout my Bachelor thesis and, in particular, came up with the initial idea of the map based user interface (chapter 4.6). Furthermore, I would like to thank the colleagues with whom I shared an office space, as well as Dr. Adrian Friday and Jamie Jellicoe for supporting me and creating a pleasant working environment.

Finally, I would like to thank my parents Harald Metzger and Stephanie Frech-Metzger, as well as my grandparents for their constant support.

Abstract

During the past few decades, we have witnessed the development of an increasing number of products and research projects in the area of lifelogging. These range from early works by Steve Mann and Microsoft's MyLifeBits project up to wearable cameras like the Autographer and Narrative Clip. These and other previous approaches focused on sensors worn by the user rather than sensors in the environment. However, as the author already discussed in [CMD14] wearable sensors have several shortcomings. These include blurry pictures due to the user's movements, sensors occluded by clothes and failing devices due to drained batteries or insufficient storage.
In this thesis we pursue a complementary approach. We designed and implemented a middleware for memory augmentations that can aggregate sensor data from a diverse set of sources. We focus on privacy protection and scalability due to the system's ubiquity and comprehensive use of sensors. The system therefore features a location based access control system that does not make users traceable. We lowered bandwidth requirements via distributed storing of sensor data and duplicate avoidance. In parallel, we designed an Android library to process location based information. The Android library can be used with the system designed in this bachelor thesis, as well as the display appropriation framework Tacita developed by researchers at Lancaster University.
We demonstrate the system's scalability by showing that response times increase linearly with the scale at which the system operates. Our measurements thereby included the simulation of up to 10000 recordings and 2000 sensors on one of the system's nodes. Finally, we investigated the system's financial feasibility by estimating its operational costs.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

In the last decades we have seen an increasing amount of products and research in the area of lifelogging. Steve Mann was one of the first who was recording physiological data and video streams from wearable cameras [Man97]. In Microsoft's MyLifeBits project [GBL+02], which was inspired by Vannevar Bush's vision of the Memex computer [B+45], Gorden Bell digitised vast amounts of his personal belongings and activities ranging from books, documents and CDs to web pages and instant messaging conversations [Mic]. In recent years products based on wearable cameras similar to Micorsoft's SenseCam [HWB+06] have appeared in the market [Aut] [Mem]. With the recent advent of wrist worn devices a health care trend propelled by lifelogging techniques seems set to emerge [Fit] [Nik].

Previous lifelogging approaches have focused on data capture on the user rather than using sensors in the environment. This is the case even though there has been exhaustive research on smart environments [SKB+98] [BRTF02]. MIT's Oxygen project made heavy use of sensors in the environment in order to provide technologies like indoor navigation and speech recognition ubiquitously [Mas]. GaiaOS is an operating system for smart environments which runs on top of an established operating system [RHR+01].

The main outcome of this thesis is a distributed system that collects a diverse set of data from ubiquitous sensors installed in the environment and worn by users. Therefore, the design focused heavily on privacy protection and scalability. We made use of techniques similar to privacy beacons described by Langheinrich [Lan05] and Kindberg's context authentication protocol [KZS02]. Furthermore, the system aims to be a middleware for memory augmentation since it has been designed in the context of the EU-project RECALL [Rec14]. Lastly, an evaluation shows the feasibility and performance of the system. Based on our cost estimate we outline possible business models financing the system.

A prototype has been implemented using state of the art technologies including the Django Web Framework, Apple's iBeacon standard and Google's Android SDK. Besides being a research project, it comprised all stages of software development including requirements engineering, specification, design, implementation and a performance evaluation.

Additionally, an Android library for sensing proximity to public displays as well as sensors has been designed. Due to its general design, it can be used with Apple's iBeacon standard and

similar technologies as well as different approaches like QR-Tags. It is designed in the context of this bachelor thesis and can be used with systems like Tacita [CKDL12], which has been developed at the Lancaster University as well.

This chapter paves the way for the following chapters. We describe previous work on lifelogging as well as smart environments and give an overview of the outcomes of this thesis.

This work was conducted while visiting Lancaster University.

1.2 Contribution

Wearable sensors are already widely deployed in the context of lifelogging. However, as the author already outlined in [CMD14] this approach has several short comings. Those include blurry pictures due to the user's movements, sensors occluded by cloths and failing devices due to drained batteries or insufficient storage. A complementary infrastructure approach can help to overcome these problems and provide further benefits in terms of recording quality, maintenance and cost-efficiency.

To benefit the already existing Keepsake framework [Nic14], the system was designed as middleware for memory augmentations. Keepsake is a research platform for memory augmentations developed in the course of the EU-Project RECALL. It forwards memory triggers to the user via channels like pervasive displays and smartphones. So far, its data sources are mainly the user's smartphone and social networks.

The system designed in this thesis needs to aggregate data from ubiquitous sensors and forwards them to memory augmentation systems. Having recent surveillance scandals in mind [Gre13] [Unt13] it is obvious that privacy protection is key for user acceptance. Additionally, a trust relationship between user and sensor owner needs to be established. Otherwise, users might fear that data fed into their memory augmentation has been manipulated. Furthermore, large amounts of data generated by sensors must not render the system inoperable due to high bandwidth and storage requirements.

Our theoretical design provides several means for privacy protection ranging from the encryption, deletion and filtering of recordings with for example a face blurring algorithm to the deactivation of data streams on behalf of the user. Access to recordings can be restricted to users who have been in sensor range. This can be checked by the system without disclosing the user's identity or making him/her traceable. Furthermore, a map visualisation allows users to check if their privacy preferences are met in certain areas and other privacy relevant information. Techniques lowering bandwidth and storage requirements comprise duplicate avoidance and distributed storing of sensor data. Finally, we present a trust relationship between the system's stakeholders.

A partial implementation based on the design has been conducted. Functions implemented range from the recording and aggregation of sensor data up to a sophisticated access control mechanism using a self made iBeacon broadcaster. To support future development we made extensive use of modularisation, and already implemented APIs and components going beyond the system's current functionalities. Finally, using the implementation we could investigate the system's scalability.

1.3 Structure of this Thesis

This thesis has the following structure:

Chapter 2 - Requirements describes a set of distinctive requirements based on a set of scenarios.

Chapter 3 - Technologies describes all technologies important for this thesis including Apple's iBeacon standard and the web framework Django.

Chapter 4 - Design presents the design of the overall system, discusses design considerations and describes core concepts.

Chapter 5 - Implementation describes the implementation of a prototype, which shows the system's feasibility and paves the way for future development.

Chapter 6 - Evaluation is composed of a performance and a feasibility analysis.

Chapter 7 - Related Work analyses previous works about smart environments, memory augmentations and privacy related topics.

Chapter 8 - Conclusion underlines outcomes of the design and evaluation phase.

2 Requirements

2.1 Overview

This chapter focuses on the requirements for the system discussed in this bachelor thesis. Firstly we present a set of scenarios illustrating the future system, its features and how it will be used. Secondly, we outline key requirements drawn from these scenarios. The requirements will then function as a foundation for the system's design.

2.2 Scenarios

The following scenarios illustrate the system's functions and how it will be used. Our aim was to come up with a diverse set of scenarios that outline the system's behaviour in different situations. Hence, each scenario falls into one of the following categories: public; semi-public; private; and involves either a single person or many people.

2.2.1 Perfect Recall

Ian is a journalist and wants to write an article about a conference, but he can't recall it very well. His visit included various talks and exhibits as well as chats with academics and company representatives. Since his e-memory used locally installed cameras to record his visit he is now able to review the conference. Furthermore, his e-memory recommends certain parts of the conference based on his level of arousal during the recording. While writing the article, he can have a look at exhibits from different angles since more than one camera was installed in the room. Due to his e-memory, he can write the article quickly and inform his readers in rich detail.

Memory augmentations will enable us to retrieve past events in great detail. Ian can replay arbitrary conversations without worrying about the technology at the time of recording. Moreover, even conversations which only turn out to be interesting in retrospect are available. Furthermore, we often have to focus on many things at once and shift the focus after short periods. E-Memory will lessen this burden by giving us the certainty to be able to re-experience situations and retrieve details we did not pay attention to. Therefore, we can deliberately focus on fewer and more important things over a long time.

2.2.2 Privacy Awareness

Layla and Killian are visiting the zoo. As well as spending time together, this is a great opportunity to take pictures of them and the animals. Clearly visible signage in front of the entrance indicates that various types of recordings are allowed in large parts of the zoo. Furthermore, video recordings are allowed in all public areas. While strolling through the zoo their memory augmentation collects pictures from various cameras worn by them, people nearby and installed locally. As they arrive at the bird house a sign and their mobile phones inform them that additional audio recordings are allowed inside. Being warned, they decide to continue their private conversation and visit the bird house later on.
At the end of the day, Layla and Killian have a wonderful keepsake of their visit. Moreover, they were aware of restrictions to their privacy at all time.

Sensor infrastructures must strengthen our privacy. Layla and Killian are able to choose if they want to enter an area in which recordings are allowed. Moreover, they are aware of recordings and can be careful to not disclose private information. Furthermore, our desire for recordings and need for privacy can be contrary. An infrastructure as described above, guarantees that our desire for recordings in certain settings is satisfied. However, it enables us to protect our privacy at the same time by deliberately avoiding certain areas.

2.2.3 Scores

Frank, Colin and Julian are arguing about the scores of their recent video game evening. Since they can't remember well, Frank starts using his e-memory. It retrieves the scores from Colin's video game console and displays them. While figuring out that Julian has won a lot of games, they have great fun watching small recaps of each game alongside a recording of them playing the game. Reminded of the terrific evening, they make an appointment for another match.

E-memories in combination with ubiquitous sensors will help us to maintain and deepen our friendships. Reminded of the great evening, Frank, Colin and Julian schedule a new appointment. No matter if we meet at home, in a cafe, or or by chance in the city centre such systems will keep track of our relationships. Therefore, they will improve our quality of life by reminding us to spend time with people we like. Furthermore, a ubiquitous system, which records gatherings of our loved ones, allows us to keep precious moments even when we do not carry a camera.

2.2.4 Password

Dorothee takes part in a meeting. All participants are wearing cameras connected to their memory augmentations. However, the password securing Dorothee's computer has been captured by one of

those cameras. Before the meeting, Dorothee has set her privacy settings to allow others' memory augmentations to capture her during the meeting, given that they grant her control over those recordings. Furthermore, her memory augmentation is running a service which detects password entries in those recordings, deletes them and eventually reminds her to change the password. Dorothee's computer is still secure and all participants can recap the meeting at home.

Memory augmentations will capture sensitive information ranging from passwords, corporate secrets, to political opinions and embarrassing moments. Therefore, being able to claim control over recordings related to oneself is crucial. Control over deleting them is essential in this. In this scenario, all participants have an interest in recording the meeting. Therefore, Dorothee uses this interest to persuade other participants to let her access their recordings. Eventually, she can run a program which warns her about captured passwords and deletes the recording.

2.2.5 Feedback

The Scottish Kickers are one of the best football teams, but they lost the last game against Edinburgh United. By combining the team player's e-memories of the game, they are able to review the game in great detail. Subsequently, they enhance their formation based on recordings from various angles, ranging from the perspectives of each player and CCTVs installed in the stadium up to a tremendous number of viewers. Furthermore, based on a combination of bio data and actions a player took, they recognise weaknesses in their training schedule and do further improvements on their strategy. Eventually, they celebrate a great success after the next match against Edinburgh United.

Detailed feedback is one of the most helpful ways to improve oneself. Recordings from several perspectives reveal weaknesses in strategy and cooperation within the team as well as the opponent's strategy. Moreover, detailed data ranging from video material, bio data and relative positions of players and the ball provide a means for improvement beyond ordinary methods. A diverse set of video recordings enables doctors to recognise slightest movements that strain joints and therefore are unhealthy for the player. Even amateur sport clubs or dancing schools can use systems reminding them of room for improvement based on their last training session.

2.2.6 Practicing

John wants to play the piano. Practicing the piano is a great pleasure for him, but he can't bring himself to practice regularly. However, his e-memory system shows him pictures of the last practice session if he is bored or procrastinating. John is reminded of his aim to play the piano and how much fun he had the last time by looking at those pictures. Therefore, he sits down at the piano and starts practicing his favourite piano pieces. By using his e-memory John learns how to play

the piano. Furthermore, he is less often bored and reaches more of his goals.

Procrastinating is a problem for many students. John has problems motivating himself to practice even though he wants to play the piano. A system, which can learn about the user via long-term recordings, can help us to change our behaviour. Due to the ubiquity of such a system it can detect unintended behaviour and draw the user's attention to it just in time. In this example, John is reminded of his aim to play the piano as he is procrastinating. This stands in contrast to today's approaches, such as to-do lists, which have a very restricted ability to reach the user proactively.

2.3 Requirements

The previous section has illustrated the future system. The requirements shown in this section are based on these scenarios and used to motivate the design.

R1 Retain recordings

This work focuses on gathering data for memory augmentations. As seen in the scenarios "Perfect recall" and "Feedback" above it is not always clear if a certain recording will be needed. Therefore, the system needs to retain them in case the user or a service acting on behalf of the user shows an interest in them.

R2 Support various data sources

Different data sources ranging from mobile and static sensors up to game consoles are described by the scenarios above. The system has to utilise them and therefore comply with their different characteristics. Mobile sensors might be restricted in terms of connectivity and bandwidth due to their mobile nature. In contrast, static sensors and data stores like game consoles might have high requirements on bandwidth especially when it comes to HD video recordings. Besides that, the system must function with various data formats, which might not be known at the time of its deployment.

All data sources must still fulfil their initial purposes. CCTV operators are unlikely to be willing to connect their cameras to the system if they can't use them for surveillance anymore.

R3 Diminish privacy concerns

Ubiquitous data capture easily awakes Orwellian visions. Stakeholders like system maintainers and infrastructure providers must not be able to invade people's privacy. Especially monitoring large areas or certain people via the system should not be possible. This includes tracking the location of users, monitoring their usage of the system or identifying them on recordings. Accordingly, we state as a requirement:

R3.1 Protect the system from being turned into a comprehensive surveillance tool

Unnecessary disclosure to other users also gives rise to privacy concerns. As seen in the scenario "Password", critical information like passwords might be recorded by chance. Therefore, the system will:

R3.2 Support filtering and deletion of recordings before they are disseminated

Deleting or filtering recordings in hindsight is one way to avoid data spills, evading recordings at all is another. In the scenario "Privacy awareness" Layla and Killian are able to avoid audio recordings intentionally since the system keeps them informed about ongoing recordings. We derive the following requirement:

R3.3 Forward privacy relevant information to the user

People's privacy needs change with the context. For instance, in the scenario "Privacy awareness" Layla and Killian don't want their private conversation to be recorded. By contrast, it is likely that meeting participants want to record discussions for later review. Therefore, we state as requirement:

R3.4 The system respects different privacy needs in different situations

R4 Scalability

The system has to serve a vast amount of users and stakeholders. The scenarios above illustrate that it aims to support a very broad range of applications starting with business and health applications up to daily use. Therefore, it is to be expected that it will attract a great portion of the population. Furthermore, sensors will cover large portions of our environment and serve resource intensive data like video recordings to the system.

R5 Responsiveness

The system has to provide nearly instant access to memories. In the scenario "Scores" Frank Colin and Julian are spontaneously accessing game stats. A lengthy loading process would render the system useless since the conversation might shift to another topic quickly.

R6 Interface to third party systems

As illustrated in the scenarios above, the system does not only have to interface with memory augmentations. Systems analysing recordings or acting on behalf of the user can be of great value. These include services protecting the user's privacy or providing feedback like the examples shown in the scenarios "Password", "Feedback" and "Practicing".

R7 Allow user to share and join their memories

Sharing content with others is common place nowadays. Everyday devices like smartphones and services like Facebook [1] and WhatsApp [2] feature corresponding functions. Some services are even specialised on sharing content like pictures and video recordings [Fac] [Cou] [Vin], which is similar to the one aggregated by the system discussed in this thesis. This shows that users have a strong interest in being able to share recordings served by the system. This is underlined by the scenario "Feedback" which shows how users can gain deeper insights by sharing and joining their e-memories.

2.4 Operational Environment

Requirements on the operational environment are given by the Lancaster University and the already existing ecosystem of applications within the RECALL project [Rec14]. This includes the programming language and web framework used to build the prototype and the operating system on personal devices.

Python The web service prototype needed to be built with the programming language Python[3]. Python is one of the most popular programming languages [Cas14] and features a vast set of mature libraries, frameworks and other extensions. Due to its popularity, it has a vibrant developer community providing a rich source of documentation and solutions to common problems.

Django Framework A further requirement on the web service prototype was to build it with the Django Web Framework [Djac]. Django aims to support rapid development and brings other amenities like a built-in ORM and an automatically generated administration interface. This suits the project well since the time given to develop the prototype was highly restricted. Django also has a vibrant community providing valuable documentation and various extensions.

[1] https://www.facebook.com/
[2] http://www.whatsapp.com/
[3] https://www.python.org/

Furthermore, the fact that Django is well known among the research group, which will use and maintain the future software underlined this requirement.

Integration with the existing ecosystem The software developed in this bachelor thesis stands in the context of a bigger project. Therefore the system should integrate with the already existing ecosystem of applications. At the time this bachelor thesis took place, the ecosystem consisted of a memory augmentation research prototype called Keepsake [Nic14], which has been developed at the University of Stuttgart. Keepsake's key components include an Android application. In order to be able to run both systems in concert, mobile applications developed in the course of this bachelor thesis should run on Android smartphones as well.

2.5 Summary

In this chapter we outlined the system's functionalities and derived properties it has to feature. A set of scenarios has outlined the system's functions and how users will use it. The scenarios aimed to be a diverse set and ranged from health and business applications to casual users using it in their leisure time. Motivated by those scenarios we subsequently described requirements for the future system. Those requirements were distinctive for such a system in the context of memory augmentations rather than an exhaustive list. The system should, for example, gather data via a wealth of ubiquitous sensors and protect users' privacy at the same time. Furthermore, it should scale well since it is targeting a large portion of the population as a user group.

Lastly, we described the operational environment of the future software. The prototype has to be built with the Python programming language and the Django Web Framework. Both have a vibrant community providing a rich source of documentation, problem solutions and matured software components. Furthermore, researchers at the Lancaster University are familiar with both and will therefore be able to run and maintain the system easily.

The next steps are to design and implement the system. Decisions made in both steps will be motivated by the requirements stated in this chapter.

3 Technologies

3.1 Overview

This chapter will describe core technologies. First we describe Django, a Python based web framework that has been used to build the prototype's web servers. We used two additional frameworks to extend Django's capabilities. Firstly, a framework focusing on REST APIs and secondly GeoDjango that allows us to handle geographical data. Subsequently, we discuss Apple's Bluetooth Low Energy based iBeacon standard, the standard Linux Bluetooth stack BlueZ, and the Raspberry Pi. Together these were used to build one of the system's characteristic features, the access control mechanism in an unobtrusive, privacy aware and power saving way. Finally, we outline the display appropriation system Tacita, which will be able to use the Android library designed in this thesis as well.

3.2 Django

Django is a Web Framework for the Python programming language. It aims for "automating as much as possible" and it "encourages rapid development and clean, pragmatic design" [Djac]. It suits projects like this with highly restricted development time very well. It features techniques, to ease and speed up development. One of them is an Object Relational Mapper (ORM), to take the burden of maintaining database tables from the user. Furthermore, the developer does not have to query the database directly. He/she, for example, does not necessarily need to write SQL queries if a SQL database is is used. This can save effort during development and eliminate an error source. Another technique is the automatically generation of the administration interface which is described in more detail below. Both are in line with Django's general adherence to the don't repeat yourself (DRY) software engineering principle. Due to its popularity and active community, Django comes with a wealth of software components and extensions. The website Django Packages, for example, currently lists over 2000 such components ranging from whole frameworks and custom data models to smaller libraries [Dan].

Django is used by a diverse set of companies. This includes companies focused on software engineering and web development:

Mozilla Mozilla is known for developing the very popular Firefox web browser and Thunderbird mail client. Mozilla uses Django during web development, for example, for the Firefox Marketplace [McK13] [Moz].

Bitbucket Bitbucket provides repositories using common revision control systems. It is one of Atlassian's products, a company focusing heavily on software development and project management tools. It currently serves 35000 customers including NASA and Audi [Djab] [Atl].

Disqus Disqus hosts comments on web blogs. Its Django based website has to deal with heavy load since it has "nearly 500 million unique visitors every month" [Dis13] [Dis11].

3.2.1 The Django Admin Site

The admin site is one of Django's distinctive features. Using the data model, Django automatically generates an administrative interface developers can customise by modifying underlying classes and functions. Since a large part of the UI's code basis is automatically generated, developers don't need to handcraft it. This can help to save development time and eliminate malfunctions caused by programming errors. Furthermore, the admin site comes with a consistent and straightforward appearance and therefore supports the developer in the UI design process. These features enabled us to produce an administrative interface in a short amount of time. It featured the intended workflows and came with a built-in user authentication.

3.2.2 The Django REST Framework

This component extends Django with functions necessary to build a REST API. These include a browsable API, integration with Django's user authentication and a test framework.

The architecture designed in this bachelor thesis makes heavy use of machine-to-machine communication. Using the Django REST framework we were able built a corresponding REST-API along with a rich set of test cases.

3.2.3 GeoDjango

GeoDjango equips Django with the capabilities to handle and process geographic data. It integrates with Django's ORM and adds spatial queries, for example, for intersections, containment and distance. Further features include the definition and display of polygons on top of maps in the administration user interface.

The system designed in this bachelor thesis has to handle geographic data at several points. Devices and privacy relevant information are assigned to certain areas. Users can check if their privacy preferences are violated in areas they reside and on paths they take. Once again by using this component, we could provide important and complex features in reasonable time.

3.3 Wireless Short Range Communication

Wireless short range communication allowed us to implement an access control mechanism (see chapter 4.3) that does not require user intervention.

3.3.1 Bluetooth Low Energy

The system's design requires mobile devices like smartphones to constantly listen for transmitters. This can easily drain the battery and therefore affect the practicality of the system. Therefore, we chose to use Bluetooth Low Energy (BLE), which targets a significantly reduced power consumption [Blu].

3.3.2 iBeacon

iBeacon is an Apple trademark and an indoor navigation standard based on BLE, which is discussed above. It involves transmitters broadcasting unique identifiers that indicate the receiver's position or context. We could use iBeacons to disseminate information to users in certain areas. The iBeacon standard constitutes a one way communication channel from the transmitter to the receiver and therefore does not make receivers traceable.

3.3.3 BlueZ

BlueZ is the Linux standard bluetooth stack. It supports BLE since version 5.0 [Pad13] and is therefore compatible with the iBeacon standard. It allowed us to build our own iBeacon transmitters as discussed in chapter 5.5.1.

3.3.4 Radius Networks iBeacon Library

On the receiving end, we used the Radius Network iBeacon library [You]. Using this open source library we were able to receive iBeacons on Android smartphones. By contrast, other libraries only allowed us to receive identifiers specific to a certain product.

3.3.5 Raspberry Pi

We used Raspberry Pis as basis for the self-made iBeacon transmitters. The Raspberry Pi is a credit card sized computer equipped with an ARM CPU running at 700MHz, 512MB RAM and two USB ports. Via the USB ports we were able to add a Bluetooth as well as a WiFi adapter that are connecting the Raspberry Pi to the rest of the system. It is capable of running a fully fledged Linux and therefore the BlueZ bluetooth stack discussed above. Furthermore, its hardware resource are sufficient to run the Django iBeacon server, which is discussed in chapter 5.5.1.

3.4 Tacita

Tacita is a framework for display appropriation developed at the Lancaster University [DLC+14]. Using their smartphones, it allows users to select content on nearby public displays. To achieve this, different localisation techniques are used to determine the closest display. Furthermore, a concept similar to our presence token (see chapter 4.3) is used to prove that a user is in front of a display. In the course of this thesis we will design an Android library, supporting localisation as well as presence token for Tacita and our system.

3.5 Summary

In this chapter we described key technologies used in the course of this bachelor thesis. Firstly, we described Django, which was used to build the prototype's majority of components. Due to its adherence to rapid development and the don't repeat your self paradigm, it is well-suited for development in a research project with a restricted amount of time. Thanks to its auto generated administration interface, we could build in a short amount of time a consistent and well structured user interface implementing the desired workflows. Django's built in ORM is a further convenience feature, which allowed us to maintain database tables easily and query them via Python code. Subsequently, we discussed Bluetooth based technologies as well as the credit card sized computer Raspberry Pi, which enabled us to implement a key feature in an unintrusive and power saving way. Finally, we outlined the display appropriation framework Tacita. The design of an Android library for this and our system will be part of the following section.

4 Design

4.1 Overview

In the first part of this chapter we present an overall architecture of the system based on the requirements stated in chapter 2.3. We motivate and discuss key concepts as well as components, their functionalities and connections to each other. Finally, we describe a trust relationship between the stakeholders and discuss how it is supported by the system. In the second part of this chapter we discuss the design of an Android library that processes location-oriented data for the display appropriation framework Tacita and our system. We present the library's overall architecture and subsequently discuss its components and their relationships. This chapter will be the foundation for the implementation discussed in the following chapter.

4.2 Architecture and System Overview

The system is a middleware for memory augmentations that aggregates data from a diverse set of sources including fixed infrastructure and mobile sensors. For instance, sensor owners can attach already installed fixed infrastructure sensors to it. However, sensors like CCTV cameras must still fulfil their initial purpose. Finally, the middleware must provide aggregated data to memory augmentations and third party services built on top of the system. Furthermore, the system provides means to avoid recordings or delete and filter them at a later point in time in order to protect the privacy of users and bystanders.

An architecture based on the requirements stated in chapter 2 is shown in fig 4.1. At the highest level the system consists of five different components, whereby only the central storage component provides an interface for memory augmentations. User interaction takes place either with the memory augmentation, with the maps component or the user's personal device. Those component's relationships and the central concepts of presence tokens and privacy preferences are described in the following sections.

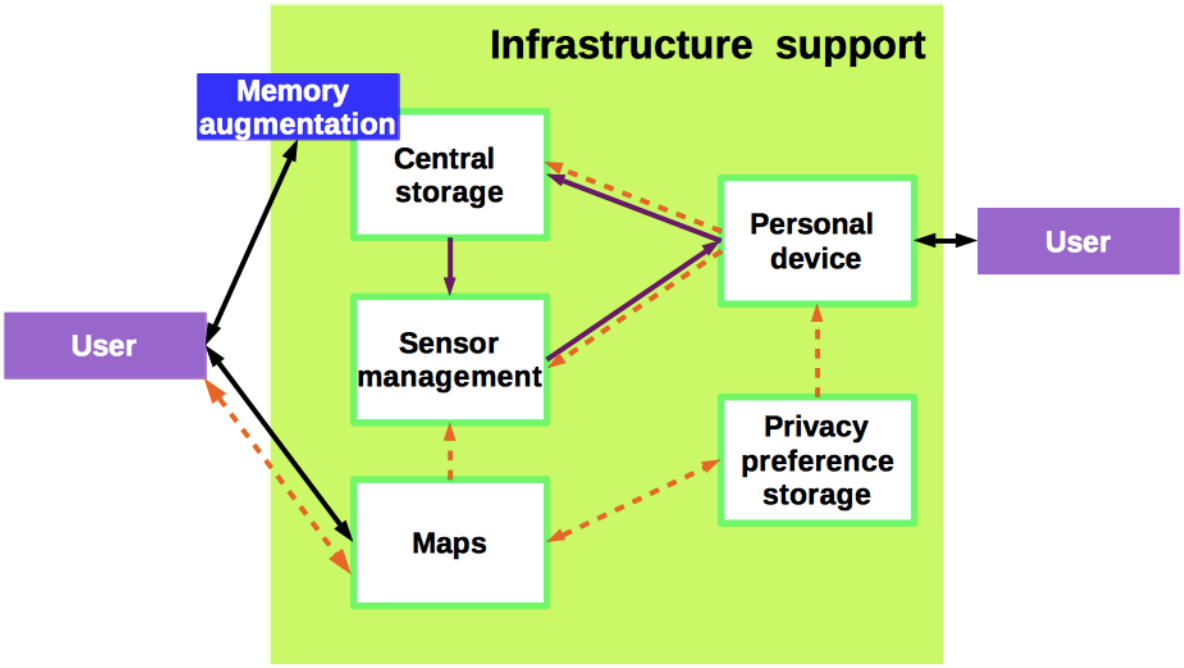

Figure 4.1: This figure shows the system's architecture. Green components are comprised by the system discussed in this thesis. The blue component represents memory augmentations using the system. The purple box represents users. Orange arrows show paths over which privacy preferences are transferred. Purple arrows show paths over which presence tokens are transferred.

4.3 Presence Tokens

Ubiquitous recording easily triggers Orwellian visions. Therefore, it must not be possible to turn the system into a comprehensive surveillance tool (R3.1). To achieve this we use presence tokens as a means to restrict access to recordings to people potentially featured by them. This is possible since holding a presence token proves that a person has been in a certain area at a certain point in time. Before access to a recording under access control is granted, the requester must prove that he has been in the recorded area while the recording has been taken by showing the corresponding presence token. This mechanism based on the context authentication protocol described by Kindberg [KZS02] will be explained in more detail in the following section.

Figure 4.2 illustrates the concept of presence tokens. Each coloured area has an unique presence token at any point in time, which can't be captured by persons in neighbouring rooms. An area must at least cover the range of sensors in it. Otherwise, people recorded by these sensors might not have access to recordings featuring them. This would, for instance, prevent them from requesting the filtering or deletion of these recordings and therefore violate

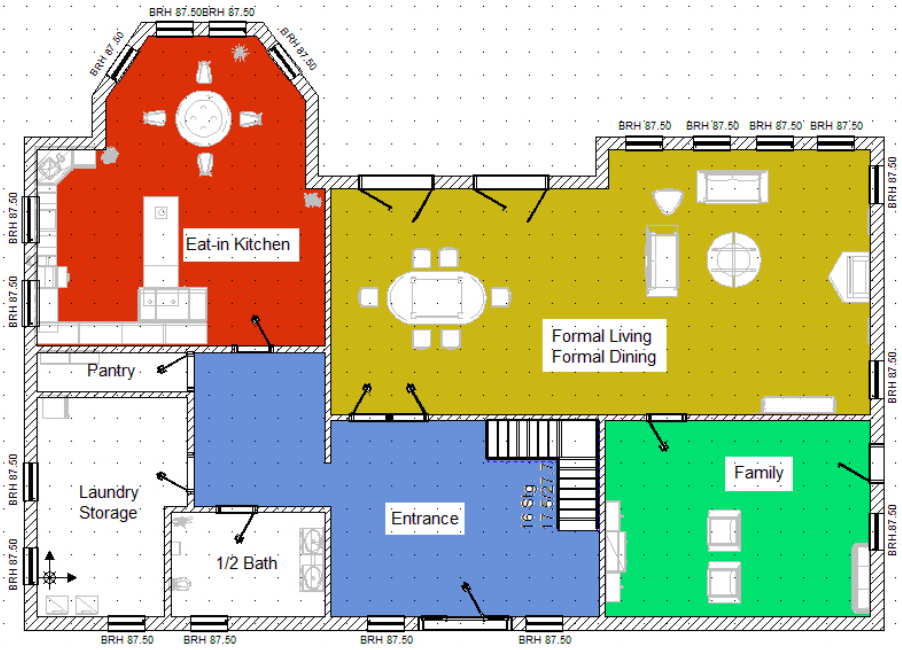

Figure 4.2: This figure illustrates the concept of presence tokens. A unique and randomly generated presence token is assigned to each coloured area. Areas with different colours have different presence tokens.

requirements R3 and R3.2. In settings like office buildings it must not be possible to spy on coworkers in other rooms. Therefore, the area needs to cover no more than the range of vision people in that area have. In other settings like lecture theatres however the area might need to go beyond what people normally can see or hear so they have a better sight of the lecturer.

Figure 4.2 discussed above aims to clarify the concept and is therefore idealised. Presence tokens are communicated via broadcasters to personal devices carried by the user. Broadcasting devices might be radio based and therefore not cover a confined area as the figure might indicate. These broadcasting devices, which don't cover a confined area can be appropriate for cases in which the range of sensors or a person's perception is not constrained as well. For instance, a person might be able to see or hear others through open doors or windows and so does he/she if his personal device picks up a broadcaster's signal coming through that door or penetrating a weakly dampening window. Similarly, a person sitting next to a wall might be able to hear people in the neighbouring room and so does he/she if his personal device picks up a broadcasters signal penetrating the wall. However, he/she can't hear people two rooms down since the acoustic waves can't penetrate two walls and neither can the radio signal originating in the room. Our system though does not reflect the degradation of acoustic waves penetrating walls or visual covers. Future work could investigate on the degradation of delivered recordings based on the signal's strength with which a presence token has been received. On the other hand, settings like meeting rooms might have a high security demand making the radio-based

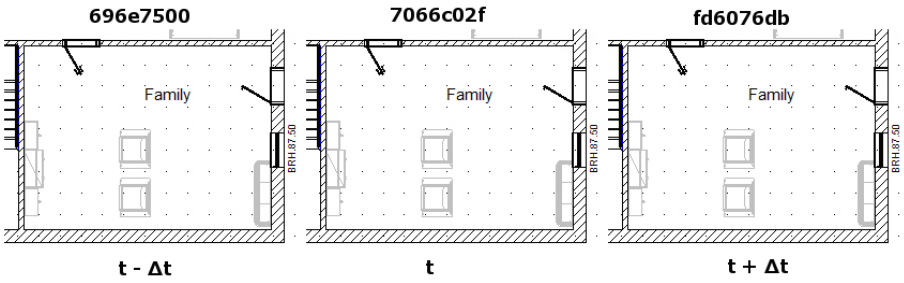

Figure 4.3: This figure shows from left to right the same room with changing presence tokens. The texts above the rooms are example presence tokens. The time is shown below the rooms.

broadcasts discussed above inappropriate. To prevent eavesdropping presence tokens could be disseminated via a cable e.g. in combination with a docking station for the user's personal device, very short range radio or copying from local displays as suggested in [DLC+14].

As illustrated in figure 4.3 presence tokens change periodically. From left to right the figure shows the changing presence token above the room. The presence tokens shown are examples with the aim to clarify the concept and not to put any restrictions on their implementation e.g. in terms of length and format. As indicated in the lower part of the figure, presence tokens change with the frequency $1/\Delta t$, which is specific to an area. As a result, users leaving an area do not have access to subsequent recordings in it since they don't receive the necessary subsequent presence tokens. However, if a user leaves an area he/she has access to recordings of that area until the current presence token changes. Therefore, during a certain timeframe the user has access to recordings of that area even though he/she is not in it. This timeframe starts with the user leaving the area and ends with the change of the current presence token. This timeframe's size can be decreased by increasing the frequency with which presence tokens change. However, this is not appropriate in all situations, for instance if the presence token is disseminated via QR-Tags and has to be scanned manually by the user. Furthermore, frequently changing presence tokens may also result in greater requirements on the reliability with which they are received and the battery consumption on the receiving end. Therefore, the frequency providing a maximum of privacy and convenience will vary with the context.

Obviously, a statically installed device that forwards presence tokens to an attacker could subvert our access control mechanism. Our aim however was to prevent the system from being turned in to a comprehensive surveillance tool. An attacker would need to use a significant number of forwarding devices in order to achieve this. Therefore, we consider this mechanism to be sufficiently secure due to the high costs of an attack.

Presence tokens allow the implementation of a location based access control mechanism that does not make users traceable. The central storage component stores presence tokens on behalf of the user and must be able to link presence tokens to a sensor management

component. However, the central storage can't link presence tokens to a certain location since a sensor management component is likely to maintain several areas (see figure 4.2). The sensor management component in turn can't link a request containing a presence token to a user since these requests are sent by central storage components on behalf of several users. Communication paths over which presence tokens are transferred are depicted in figure 4.1. A similar approach preventing public display owners from creating user profiles is described in [DLC+14].

4.4 Privacy Preference

A means for users to express their privacy preferences is important for ensuring respect and trust between users and service providers (R3). Using privacy preferences, users can state which kind of recordings they are willing to accept. A user might be indifferent about video recordings but does not want to have audio recordings in his/her vicinity. Privacy preferences can also depend on place and time (R3.4). For instance, a users can state that they do not want to be videotaped on their daily commute to work. Besides that, users can issue the filtering or deletion of recordings (R3.2) or put conditions on their dissemination. For instance, users might be willing to share recordings taken by their personal device upon the condition that others share their recordings as well.

A significant volume of existing work has been conducted around the use of machine readable languages used to communicate privacy preferences and policies [CLM02a] [CLM+02b] [EFH+97] [KKH+05] [MFD03]. Further research on the design of such a language is not included here as it is beyond the scope of this bachelor thesis.

4.5 Central Storage

Recordings aggregated via wearable and fixed infrastructure sensors (R2) are made available by the central storage (R1). Connected memory augmentations and third party systems can access those recordings on behalf of a user, for further processing (R6). However, access is only granted if the user holds the corresponding presence token (R3.1) or if another user grants access to his/her recordings (R7).

Recordings can easily consume large amounts of storage especially when it comes to videos. A naive approach of storing recordings on a per user basis would exacerbate this since many duplicates would be hold by the central storage. A solution for this is to store recordings per presence token rather than per user. This requires the central storage to hold a user's presence tokens in order to check if he/she has access to a recording. However, we expect presence tokens to consume far less storage than recordings on average. This is due to the fact that a portion of the recordings will be storage consuming audio and video recordings. A further

benefit of this optimisation is a decreased bandwidth demand. The central storage might need to download recordings from a sensor management component before it can serve them. Using the technique discussed above it can serve a downloaded recording to an arbitrary amount of users without the need to download it again. This technique works only for recordings served by sensor management components. However, recordings directly fed in to the central storage by, for instance, wearable sensors and other personal devices need to be stored only once as well. In this case the system only needs to keep track of to whom access permissions have been granted.

Recordings stored by the central storage can be encrypted. Those recordings may not be decrypted by the central storage if the user withholds their decryption key. This can be advantageous if the provider of the central storage is not fully trusted. For instance, if recordings reveal company secrets, passwords or private information, which might negatively impact a person's private or professional life (R3 and R3.4). However, this has certain drawbacks conflicting with requirements R4 and R5 and should therefore not be the general case. In order to process such a recording the whole recording needs to be transferred to the memory augmentation, which in turn needs to decrypt and process it then. This can easily lead to high requirements on bandwidth and computational power especially when the memory augmentation needs to examine several recordings. For instance, if the user searches for a video recording by a certain feature in it, it is likely that the memory augmentation needs to download and decrypt several video recordings until the sought recording is found. Using unencrypted recordings these problems can be overcome. The central storage itself could search through recordings and therefore provide a rich query interface. This reduces bandwidth requirements drastically since only search requests and results are transferred. Furthermore, overheads arising from the repeated decryption of recordings are eliminated.

4.6 Maps

The maps component is intended to help the user to protect his/her privacy. This component provides detailed information about installed sensors and recording policies (R3.3) via a map visualisation similar to Google Maps[1] or OpenStreetMap[2]. Furthermore, the user can upload privacy preferences for selected areas and intervals (R3.4). For instance, a user can state that audio and video recordings should be deactivated on his daily commute to work. Figure 4.4 shows mockups of the map based visualisation. User can manually check privacy relevant information including the position and kind of sensors deployed, allowed and forbidden recordings, and if recordings are encrypted. Alternatively, using the UI users can perform an

[1] https://maps.google.com/
[2] http://www.openstreetmap.org

Figure 4.4: The figure shows UI mockups belonging to the maps component. Users can check privacy relevant information manually (left hand side) or automatically check if his/her privacy preferences are met (right hand side).

automatic check to see if their privacy preferences are met, for example, in a certain area or along a certain route.

Using a map visualisation has several benefits. Maps have a widespread use in combination with information technology but also a long tradition as non digital objects and can therefore be assumed as common knowledge. Hence, map visualisations provides a well known and intuitive way to interact with the system for a large and diverse set of the population. Furthermore, it provides many memory cues which are especially helpful if the user schedules his/her privacy preferences according to daily habits, which requires the knowledge of, for instance, daily taken routes and entered buildings.

Informing the user about privacy relevant details via a map can have advantages compared to other systems like privacy beacons suggested in [Lan02]. Privacy beacons require the user to be in a certain area to receive privacy relevant information. This has drawbacks in some situations e.g. a user might have chosen another route if he/she would have known that not the entire route complies with his/her privacy preferences. Our approach enables the user to check for violations of their privacy preferences ahead of time and make decisions accordingly. However, privacy beacons have clear advantages e.g. when it comes to warning users just in time. Therefore, these two approaches can be considered to be complementary.

4.7 Privacy Preference Storage

The user's privacy preferences are synchronised across different components via the privacy preference storage. The user can add, edit and delete privacy preferences via the map based visualisation (see section 4.6), which in turn forwards these changes to the privacy preference

storage. Once the privacy preference storage has updated its record of privacy preferences then personal devices can download them and update their records.

4.8 Personal Device

The personal device has a key role in privacy protection. It collects presence tokens in order to prove that a user has been in sensor range at a certain point in time. This can be done either automatically or manually e.g. via BLE for the first one or QR-Tags for the latter. Those presence tokens are then forwarded to two other components. Firstly, to a central storage which uses them to retrieve the corresponding recordings. Secondly, to a sensor management component as credentials alongside a request to delete or filter recordings.

Encryption keys securing sensitive recordings are collected by the personal device as well. This can be done in various ways ranging from wireless short range transmitters, which ensure that only people in sensor range can decrypt a recording to manual text input after user authentication to ensure that only designated people can decrypt a recording. The personal device forwards them either to a central storage component or an application trusted by the user and connected to a central storage component. The benefits and drawbacks of both alternatives have been discussed in section 4.5.

On the UI side, information about nearby recordings and guidelines are forwarded to the user (R3.3). This includes especially warnings about possible violations of the user's privacy preferences. For instance, a user who does not want to be video taped will be warned via his personal device if he is near an area in which video recordings take place. Once a user enters a recorded area he/she might want prevent a recording from being disseminated (R3.2). By providing this functionality through the personal device with its ubiquitous nature, the user can issue the filtering or deletion of the recording in situ and does not have to remember to do it at a later point in time. This in turn decreases the probability that the user forgets about it and therefore strengthens his/her privacy.

4.9 Sensor Management

4.9.1 Overview

Stationary sensors in a certain area, for instance, a floor in an office building are attached to a sensor management component. Recordings taken via these sensors are cached by the attached sensor management component from which the central storage components can download them (see section 4.5) (R1). Furthermore, sensor management components feature several mechanisms protecting user's and bystander's privacy (R3) ranging from the dissemination of presence tokens to the encryption, degradation and deletion of recordings (R3.1 and R3.2).

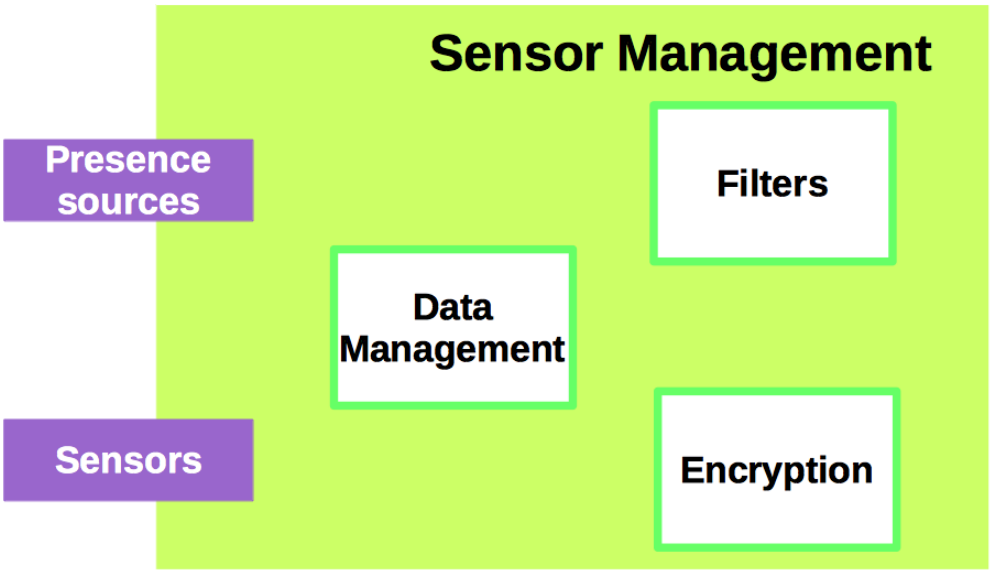

Figure 4.5: This figure shows the sensor management's subcomponents (green) and external components (purple).

This diverse set of functionalities featured by the sensor management component makes it more complex than the components discussed above. Due to that, we grouped those functions in subcomponents as shown in figure 4.5. These subcomponents are discussed in the following sections.

4.9.2 Presence Source

Before granting access to a recording, the system proves if the user has been in sensor range during the recording (R3.1). Being able to do this without disclosing the user's identity at any point in time is one of the system's distinctive functionalities (see section 4.3) (R3). To achieve this, presence sources broadcast presence tokens (see section 4.3). Examples for such devices and their deployment are wireless stations like Apple iBeacons [App] or displays showing presence tokens as QR tags [DLC+14]. The selection of a device for broadcasting tokens will be dependent on context.

4.9.3 Sensors

The sensor management's main purpose is to make the recordings of attached sensors available to central storage components (R1). Sensors range from cameras and microphones to thermometers and motion sensors (R2). However, they are not constituents of a sensor

management component but rather external parts. If sensors are already integrated into an infrastructure a sensor management component can be seen as a further data sink (R2).

4.9.4 Encryption

Encrypting recordings is a further strategy to protect user's and bystander's privacy. A central storage component aggregates recordings on behalf of a large number of users. Therefore, the operator of a central storage component may have access to a comprehensive set of recordings in private and public spaces. Encrypting recordings before they are aggregated prevents the operator from turning the system into a surveillance tool and reduces the risk of data breaches (R3.1).

4.9.5 Filters

These subcomponents are used to filter recordings before central storage components can retrieve them (R3.2). System users can, using their privacy preferences, trigger the filtering of recordings. Such filters include screen and face blurring, voice changing or quality degradation. Furthermore, this functionality can help to avoid security breaches e.g. via the blurring or deletion of recorded computer screens and passwords.

4.9.6 Data Management

A sensor management component caches and manages recordings captured by attached sensors. Central storage components can retrieve recordings from it. However, recordings can be locked for a certain amount of time. While being locked, recordings are either not delivered or are delivered in a degraded form. This gives people who have been recorded a chance to request the deletion or filtering of a recording before central storage components can retrieve the recording (R3.2). Downloadable recordings can be either publicly accessible or the requester has to proof that he/she has been in sensor range during the recording. This is possible via presence tokens as described in section 4.3.

4.10 Trust Relationship

Since we envisage memory augmentations as ubiquitous systems supporting the user on a daily basis a trust relationship between the user and sensor owner has to established. Users might be reluctant to use the system if they fear that recordings may be manipulated. Besides that, trust in the reliable functioning also plays an important role. The users may abandon the system if it fails to record important information and the resulting costs outweigh any benefit. However,

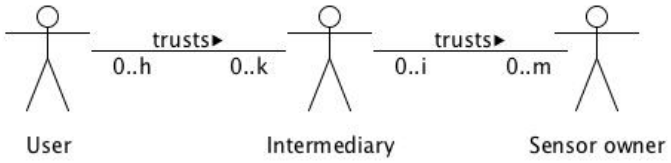

Figure 4.6: This figure illustrates the trust relationships within the system. To be of benefit k must be a much smaller number than m.

it might be impossible or too time consuming for users to check the reliability of individual sensors and sensor owners. This is particularly likely to be true if the user utilises sensors owned by multiple sensor owners on a daily basis. Therefore, we introduce the intermediary as a third role that establishes trust relationships with sensor owners. Thus the user has to trust a single or a few intermediaries instead of a wealth of sensor owners. If for instance, a sensor has an obvious malfunction the user can blame the intermediary and doesn't have to find out the sensor owner or get in touch with him/her. The intermediary in turn can then forward the complaint to the sensor owner. We did not investigate techniques enabling the intermediary to check sensors or recordings for subtle manipulations since this would go well beyond the scope of this bachelor thesis. Figure 4.6 illustrates this trust relationship. To be of benefit k must be a much smaller number than m.

Our system supports this trust relationship. The intermediary could run a service providing a list of trusted sensors to the user's personal device, the central storage and the map component. Using this list of trusted sensors the central storage could retrieve recordings only from sensor management components with trusted sensors attached or mark recordings as untrusted. Furthermore, warnings about untrusted sensors can be displayed using the personal device and the map component. Alternatively, the intermediary could run the central storage directly instead of providing a list of trusted sensors to a third party central storage.

4.11 Android Library

4.11.1 Overview

In this chapter we will describe the library targeting the system designed in thesis and the display appropriation framework Tacita described in section 3.4. This library provides a means for both systems to forward location dependent information to the user and other parts of the system. For instance, the system designed in this thesis can use the library to inform the user about nearby recordings and forward presence tokens to a central storage component.

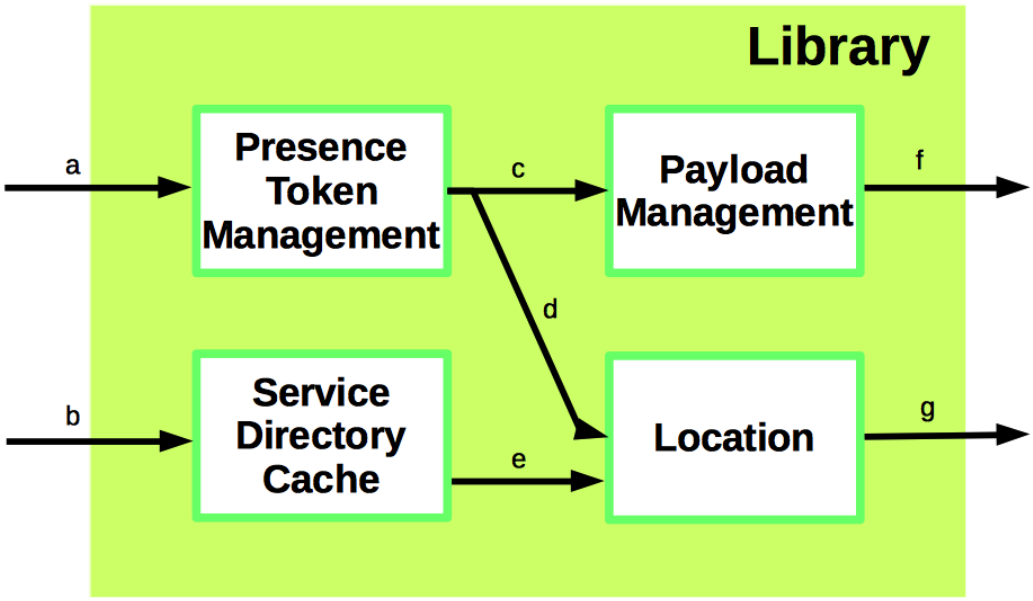

Figure 4.7: This figure shows the Android library targeting Tacita and the system designed in this thesis. It shows the library's components (green), external and internal interfaces as well as the data flow (black arrows).

In the context of Tacita, the library can be used to schedule the display of applications if a user enters the vicinity of a display. Furthermore, the user's personal device can show nearby displays along with a selection of display applications to choose from. While the sections above were discussing a new kind of system this section is describing the design of a technical component. Therefore, this section will be much more of a technical nature than the sections above. Figure 4.7 shows the library and its subcomponents. Arrows depict internal and external interfaces as well as the data flow between them. The subcomponents are discussed below. Detailed UML class and component diagrams are in Appendix A.

4.11.2 Presence Token Management

This component receives presence tokens (a) via arbitrary sources. It checks if the current presence token has changed or, if more than one has been received, if a new one is among the received presence tokens. New presence tokens are received if, for instance, the user enters the range of a presence source or the current presence token changes as described in section 4.3. New presence tokens are passed on via a callback to the payload management (c) and location subcomponent (d).

4.11.3 Service Directory Cache

This component caches information based on the user's location (b). Cached information includes nearby displays and their supported applications as well as privacy relevant information such as nearby sensors and recording policies. With this component, the mobile application can still operate if the device has lost its internet connection and the services providing these information are not available. Furthermore, the mobile application needs to display information in a timely fashion. Time consuming queries over the internet however may thwart this. A user, for instance, might not be warned early enough about recordings in an area he/she is entering if the processing of the request for these information needs too much time. With this component, the application can circumvent this and retrieve this information quickly using a cache (e).

4.11.4 Location

If this component receives a new presence token (d) it queries the cache (e) for e.g. nearby displays and sends this information to another component using a callback (g). In the context of Tacita and the system designed in this thesis, the receiver will be most likely the UI.

4.11.5 Payload Management

Once this component receives a new presence token (d) it creates a payload for an HTTP request and queues it. In the context of the system designed in this thesis, this payload contains, for instance, a presence token and will be sent to the central storage. Another component, can than pull the payload out of the queue (f) and send it. To avoid constant polling of the queue, the Payload Management component informs registered components about new entries in the queue via a callback (f).

4.12 Summary

In this chapter we presented an architecture and design based on the requirements in chapter 2. We discussed the key concepts of presence tokens and privacy preferences, which are used to protect privacy of users and bystanders. Subsequently, we discussed the system's components starting with the central storage, which aggregates recordings on behalf of the user. Those recordings are retrieved from sensor management components which are directly connected to sensors. We introduced a map based UI that enables the user to check privacy relevant information before he/she enters a recorded area. Furthermore, we use the user's personal device to realise an access control mechanism based on the constant collection of presence

tokens. Finally we motivated a trust relationship between the stakeholders and discussed how the system supports this relationship.

In the second part of this chapter we presented an Android library targeting the display appropriation framework Tacita and the system designed in this thesis. Both systems forward information to users based on their location and collect presence tokens via the personal device.

5 Implementation

5.1 Overview

In the previous chapters we were discussing requirements for the future system (chapter 2) as well as an architecture that would meet them (chapter 4). We decided to implement the system partially due to the vastness of the designed architecture and the limited amount of time. This implementation shows the feasibility of key concepts and components and provides a foundation for the preceding evaluation and future development.

Firstly, we discuss the development process as well as programming and quality assurance tools. Subsequently, we give an overview of the implemented components and their relationships, followed by a discussion of them. The implementation comprises of three Django web servers and an Android application. It is capable of recording data streams and transferring them to a central storage component. Furthermore, it features the presence token based access control mechanism, which, for instance, stops the transfer of recordings if the user leaves the sensor range. Finally, we present an exhaustive documentation of the webservers' REST-APIs.

5.2 Development Process

The development was governed by the waterfall model and comprised the following stages in the given order:

Requirements analysis Requirements were drawn from three sources; meetings with Prof. Dr. Davies at the Lancaster University and a set of scenarios approved by him. The third source was intensive literature research which included similar projects (see chapter 7).

Specification We created a specification based on the preceding requirements analysis phase. The specification describes the demands on the future system and includes a quantity structure, scenarios, functional and non-functional requirements, use-cases, and initial set of user interface mock-ups as well as a dictionary of terms. The specification builds the foundation for the succeeding phases and motivates the system's design presented in the draft. Furthermore, the use cases can be used to derive system test cases.

Draft We designed the future system based on the specification. This included the creation of several UML diagrams showing the system's architecture as well as its internal and external interfaces on different abstraction levels. On the lowest level we specified the system's classes via UML class diagrams.

Implementation and Testing Since this thesis is focusing on the design of the system both phases, implementation and test were only a total of 22 days long. In this time, we implemented three servers and one mobile application featuring distinctive system components. Furthermore, we created a broad set of module tests, which allows for regression tests during future development.

5.3 Development Tools

We used two Integrated Development Environemts (IDE) during the development. Firstly, JetBrain's free community edition of PyCharm [Jet] for all system parts implemented in Python. Secondly, Google's Android Studio [Goob] for the mobile application running on the user's personal device. We made extensive use of the following features: code completion, syntax highlighting, static code analysis and error highlighting as well as refactoring functionalities. Besides that, PyCharm offers type hinting, which allows the programmer to make types of function parameters explicit. This has proven useful since it allows autocompletion where it has not been possible before and detects bugs caused by erroneous function calls.

Mercurial [Mer] was our revision control system. It provided a means for collaboration and the ability to manage changes on documents and the source code. It has been made available to us by the Lancaster University and therefore also functioned as backup for documents and source code.

We used UMLet [UML] during design and implementation to create UML diagrams. It focuses on drawing rather than modelling functionalities and brings a minimalistic, easy to use, user interface. We made heavy use of it while creating architecture and class diagrams.

5.4 Quality Assurance

Besides the tools mentioned above we used the following for quality assurance:

Android unit tests Android test cases are JUnit based and part of the Android test framework, which includes further functions and tools. Unfortunately, Android test cases need to run on an Android device or an emulator and therefore consume a considerable amount of time.

Django unit tests Django brings its own unit tests based on the standard Python unit tests [Dja05]. Unit tests are by default written in the tests.py file which is in the root directory of a Django application. However, we organised test cases in different files for better clarity. The tester can then execute them easily via the command `./manage.py test` or, for example, `./manage.py test -pattern="./SensorManagementApp/test/testRESTAPIInfo.py"` for tests in a different file. While running tests, Django automatically uses a dedicated test database rather than the production database. This avoids unwanted modifications on the production system and takes the burden from the developer to set up a test database.

Django REST framework unit tests The Django REST framework [Djaa] extends the Django unit tests to simulate REST-API requests and assess the corresponding responses. More specifically, the developer can specify the HTTP request method, send and receive JSON objects and check for returned error codes or exceptions on the server side.

Checkstyle In combination with Android Studio we were using Checkstyle [Shi06] while developing the mobile application. It is a static code analysis tool checking for compliance with built-in programming guidelines, which help to increase readability and re-usability. Since it is targeting Java applications only we couldn't use it for the remaining Python based system parts.

Autopep8 Autopep8 [Hat] automatically formats Python code so that it conforms with the Python standard styleguide pep8 [RWC01]. This gives Python code a consistent appearance across the whole project and helps to increase the readability.

isort Isort [Cro] automatically arranges `import` statements in Python source code alphabetically and into groups.

5.5 Implemented Functionality

In the previous sections we described the requirements on the future system and its design. Based on that, we conducted an implementation, which will be described here. Due to the limited amount of time and the size of the overall system we decided to restrict the implementation to distinctive functionalities. The aim of the implementation is to:

- Show the feasibility of the designed system.
- Build the foundation for the succeeding evaluation.
- Build a foundation for future development.

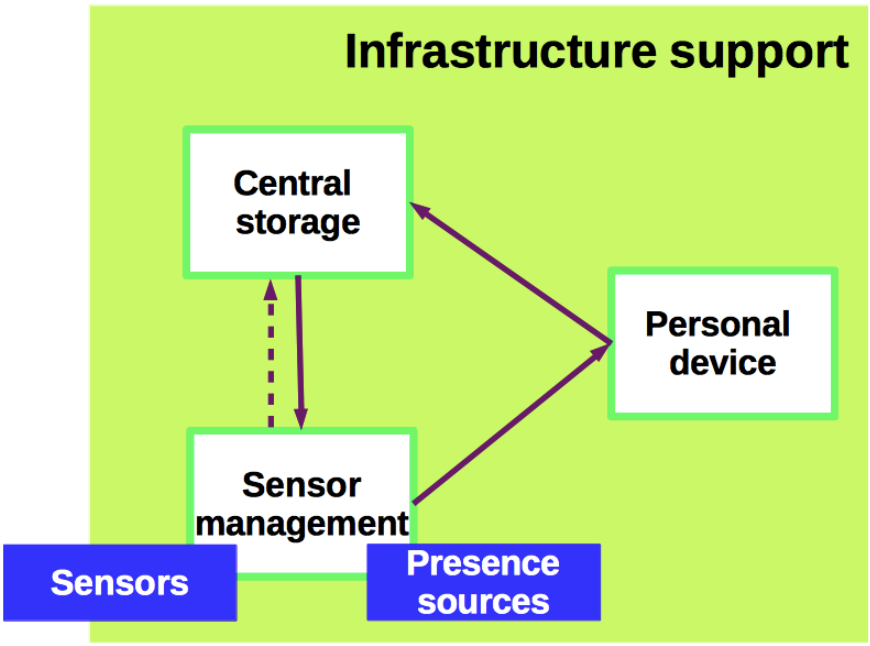

Figure 5.1: This figure shows implemented components (green) and communication paths between them. The dashed arrow depicts the transfer of recordings and solid arrows the communication of presence tokens. Sensors and presence sources are attached to the sensor management component.

In our implementation we focused on recording sensor data and their delivery to memory augmentations. Furthermore, we implemented privacy relevant components. This includes the access control mechanism based on presence tokens (see chapter 4.3). Implemented parts and the communication paths between them are depicted in figure 5.1 and are discussed in more detail below.

5.5.1 Sensor Management

The sensor management component aggregates sensor data from sensors attached and makes them available to the central storage component (see chapter 4.9). Aggregated recordings are made available via a REST API, which is designed in an asynchronous fashion in order to avoid long response times while the system is preparing the requested recoding. The following passage describes the process of requesting and downloading a recording. Firstly, in order to retrieve a recording, the central storage has to provide the corresponding presence token (see chapter 4.3). As a response, it receives a URL pointing to a dynamically generated JSON object, which contains a flag that indicates if the recording is ready for download or still in preparation. Once the recording is ready to download, this object contains a URL to

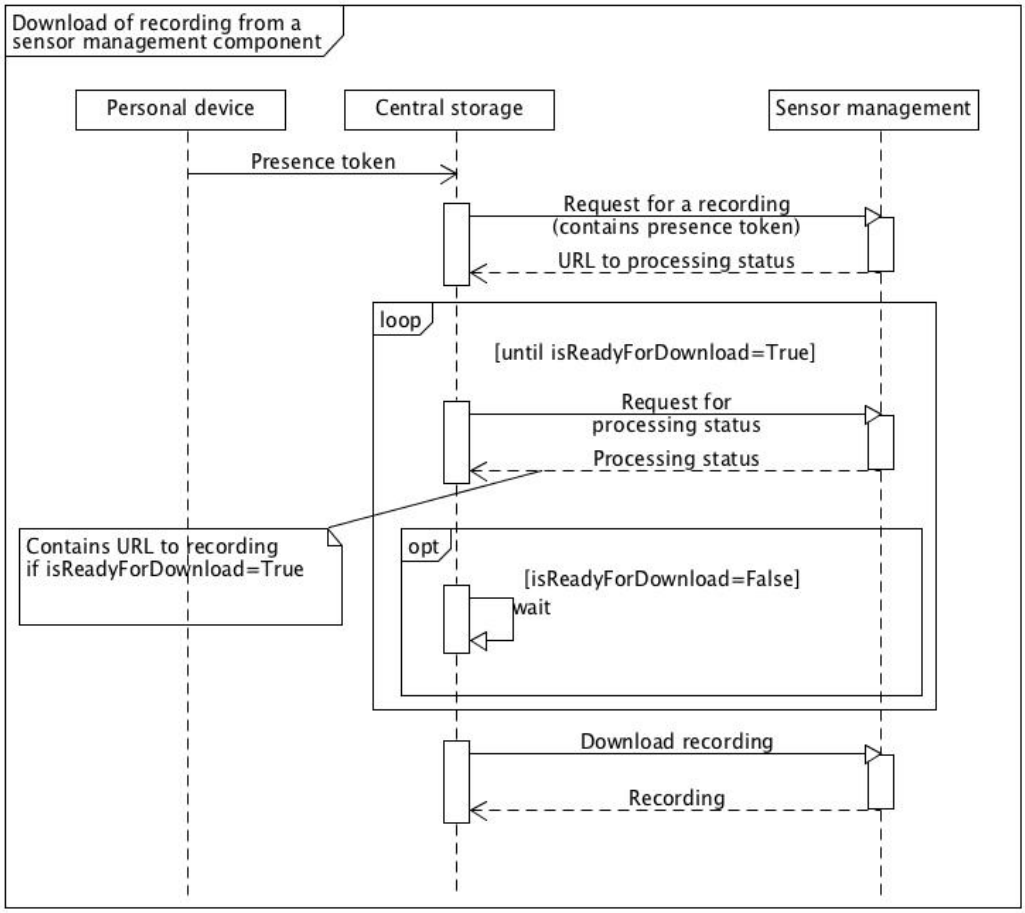

Figure 5.2: This sequence diagram shows steps necessary for a central storage component to download a recording from a sensor management component.

the downloadable archive. Recordings are shipped as tarballs in order to reduce bandwidth and storage requirements (R4). This process is illustrated by figure 5.2 and an exhaustive documentation of the REST-API can be found in chapter 5.5.4.

As described in chapter 4.9.6, recordings can be prevented from being downloadable in order to give users a chance to request their deletion or filtering. In this case the requester receives a corresponding status message and the amount of time until the recording is available.

As described in chapter 4.9.4, recordings can be shipped encrypted in order to avoid unnecessary disclosure. The command line tool OpenSSL [Ope] has been used to encrypt recordings with AES 256bit. AES is a state of the art encryption algorithm used by major applications [App14] and can therefore be assumed secure.

As stated by requirement R2, the system has to support a heterogeneous set of sensors and presence sources. Furthermore, those might be added in the future and not known at the time of this bachelor thesis. Therefore, the sensor management component features a plugin interface. Each plugin type, its functionalities, interface and an example implementation are described bellow.

Sensor plugin A sensor plugin is connected to a single, or group of sensors. It has to record sensor data for a given amount of time. Recordings are stored in the folder `raw_data`, which is located in sensor managements root folder. In-line with the requirement for extensibility discussed above, recordings can be stored in arbitrary file types. The start time of the recording, its duration and the filename to use are passed to the plugin via its constructor. As an example and in order to have a working prototype, we implemented a plugin for a camera attached to the machine running the sensor management server. During development this camera has been the webcam integrated into an Apple MacBook Air. The plugin uses OpenCV [Ope14] in order to record a video feed for the given amount of time. The recording is saved in an AVI container and uses the XVID [Xvi14] MPEG-4 codec.

Broadcaster plugin Broadcaster connected to a broadcaster plugin function either as presence source or are used to disseminate encryption keys. The presence token or encryption key and the duration they need to be broadcasted are passed to the plugin via its constructor. We implemented a broadcaster plugin for the iBeacon server discussed below. In order to save development time we used a single broadcaster for both, as presence source and to disseminate encryption keys. The underlaying iBeacon standard does not allow us to send both at the same time. Therefore, the presence token is broadcasted in the first half of the specified time and the encryption key in the second half. A device entering the range of the broadcaster needs a way to distinguish between both. We repurposed the major and minor value, which are part of the iBeacon specification and therefore part of the broadcasted data. Presence token have a major and minor number of zero. Encryption keys have a major and minor value of one.

Plugins are set up and maintained via the sensor management component's administration interface. Firstly, the user has to register plugins by providing a name, description and an area, which is either recorded or an approximation of the broadcaster's range. Subsequently, the user can group registered broadcasters and sensors by area. Recordings can be downloaded via presence tokens disseminated by broadcasters in the same area as the recording sensors. Figure 5.3 and 5.4 show the corresponding parts of the administration interface. We used Django's automatically generated administration interface and the GeoDjango framework to implement it.

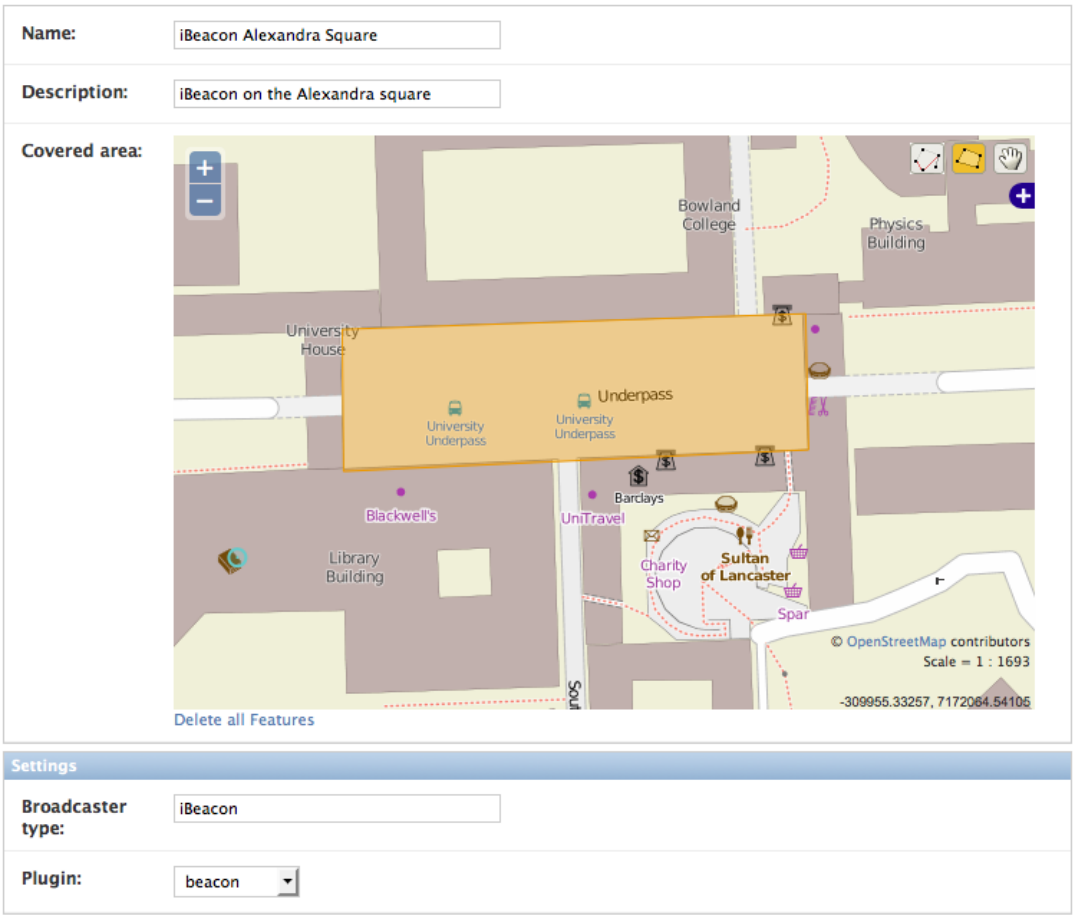

Figure 5.3: This figure shows the sensor management component's administration interface. A broadcaster can be registered using this form. Besides name, description and type, an approximation of the broadcaster range can be specified by using a map. The plugin to register can be selected at the bottom of the form.

Presence Token

Presence tokens are used to enforce access control and to unambiguously identify a requested recording. Therefore, they should be hard to guess and the probability that two different recordings have the same presence token should be very low. We used UUIDs as presence tokens. A UUID is 128bit long and therefore satisfies the requirements stated above very well.

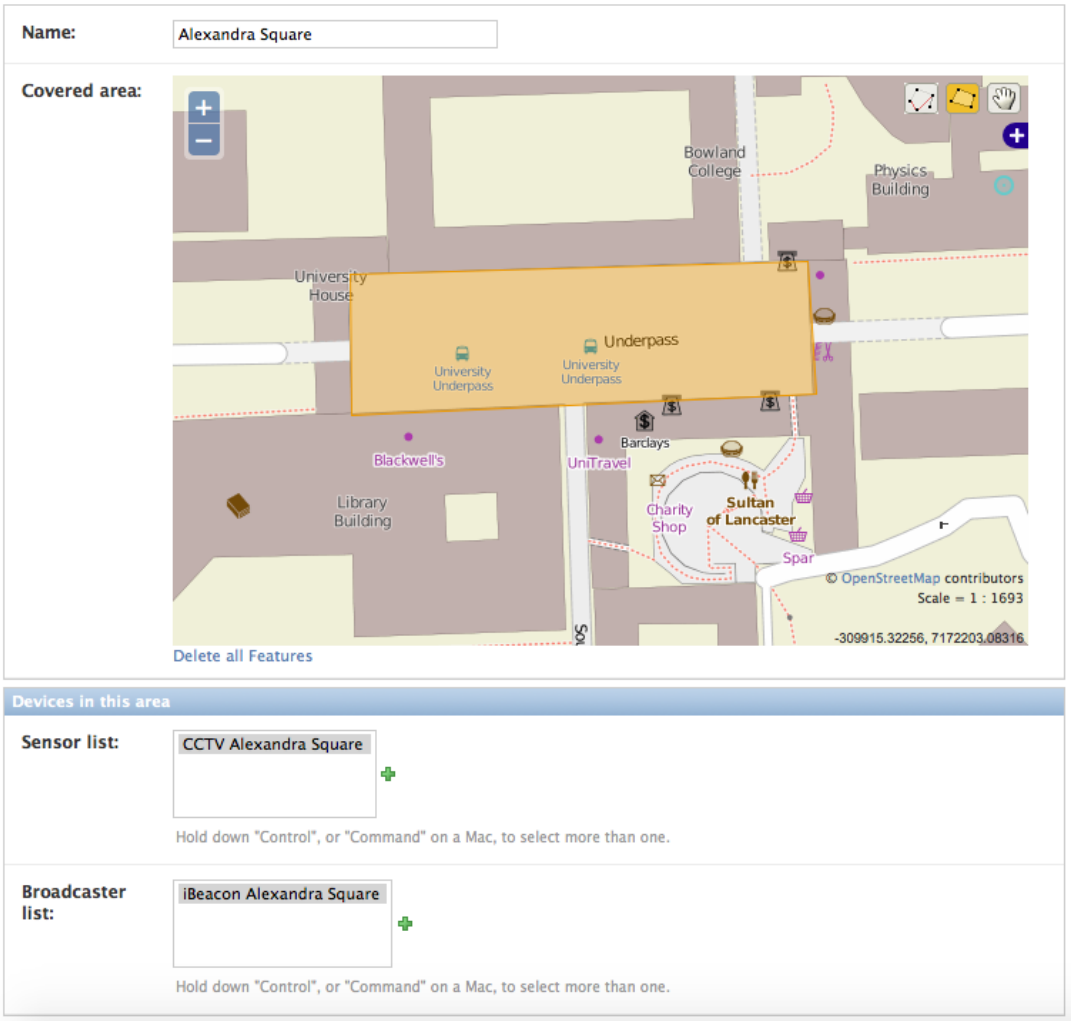

Figure 5.4: This figure shows the sensor management component's administration interface. In the upper part and middle of the form broadcasters and sensors can be grouped by area.

iBeacon Server

The iBeacon server offers a means to remotely control an iBeacon device via a REST API. This includes the following functionalities: setting the UUID, major and minor value to broadcast as well as start and stop broadcasting. In the context of this system, we used the iBeacon server to disseminate presence tokens and encryption keys. In order to broadcast via the iBeacon standard, we needed low level control of the Bluetooth device. To achieve that, we used the command line tools `hciconfig` and and `hcitool`, which are part of the standard Linux Bluetooth stack BlueZ.

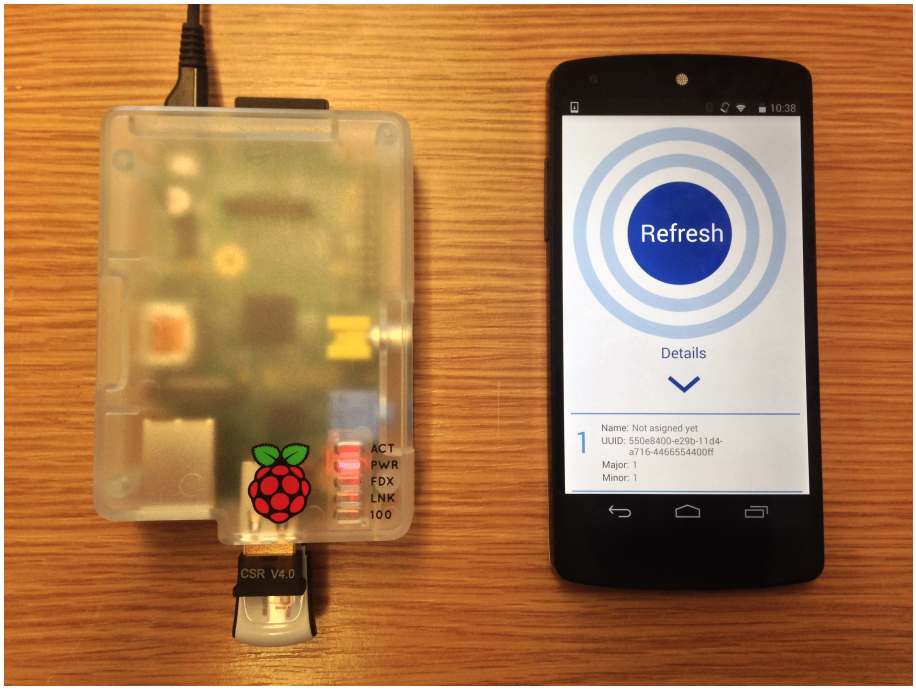

Figure 5.5: This figure shows on the left hand side, a Raspberry Pi running the iBeacon server. On the right hand side, an Android smartphone receiving a UUID broadcasted via the Raspberry Pi.

The iBeacon server has been tested under Raspbian, which is based on the Debian distribution and tailored to the Raspberry Pi. The underlaying Raspberry Pi has a CSR Bluetooth 4.0 Dongle attached, which was used to broadcast via the iBeacon standard. Also attached was a WiFi Dongle which allowed it to connect to the sensor manager wirelessly. Figure 5.5 shows a Raspberry Pi running the iBeacon server while an Android device is receiving the broadcasted UUID.

5.5.2 Central Storage

The central storage is aggregating recordings and provides them to memory augmentations (see chapter 4.5). It interfaces with personal devices and sensor management components, which are also discussed in this chapter. It features a REST-API at these connection points and a corresponding client for sensor management components. Once it receives a presence token from a personal device it starts querying the sensor manager for the corresponding recording. Downloaded recordings will be decompressed and stored in the folder `plain_data` in the application's root directory. Figure 5.2 illustrates the download process.

5.5.3 Personal Device

The personal device component is implemented as an Android application. It collects presence tokens via the iBeacon standard and forwards them to the central storage component (see chapter 4.8). Using the Radius Networks Library discussed in chapter 3.3.4, it scans periodically for iBeacon UUIDs. It checks if the received UUID is a presence token or an encryption key via the major and minor value sent with it. Complete pairs of presence tokens and encryption keys are stored in a queue. Once the device is connected to a WiFi network it dequeues these pairs and sends them to the central storage component. Pairs not reaching the central storage are queued again.

5.5.4 Rest-API

Sensor Management

Request a recording

URL	/recording/get/
Request method	POST
Description	Recordings are requested via this resource. The requester has to send a presence token encapsulated in a JSON object. As a response the system provides a URL pointing to an automatically generated JSON object, which contains a flag that indicates if the recording is ready for download or still in preparation.

Get the processing status of a request for a recording

URL	/recording/task/«presence_token»/
Request method	GET
Description	After a recording has been requested the status of its processing can be checked via this resource. This includes if the processing has failed or is still running, a human readable status message and a URL to the requested recording if it is downloadable.

Download a recording

URL	/recording/resources/«presence_token»/
Request method	GET
Description	If a recording has been successfully prepared for download, a client can download it via this URL.

Get the types of sensors attached	
URL	/info/sensor-types/
Request method	GET
Description	Responds with a list of attached sensors and their types (e.g. video, audio).

Get information about the used encryption algorithm	
URL	/info/encryption/
Request method	GET
Description	Responds with information about the algorithm used to encrypt recordings.

Get a list of filter provided	
URL	/info/filter/
Request method	GET
Description	Responds with a list of available filters. Filters are used on behalf of users to protect their privacy (e.g. face blurring).

Information about the maintainer	
URL	/info/maintainer/
Request method	GET
Description	Responds with information about the sensor management component's maintainer. This includes the maintainer's name, a description and a homepage.

Add an announced recording	
URL	/privacy-preference/announcements/add/
Request method	POST
Description	The user can announce his/her privacy preferences for a certain area. After receiving them the system checks if they conflict with the recording guidelines in that area. For instance, the user does not want to be video taped but the allowance for video recordings in that area is guaranteed. If that is the case, the system returns a list of conflicts and a corresponding HTTP status message.

Get a list of announced recordings	
URL	/privacy-preference/announcements/get/
Request method	POST
Description	Responds with a list of announced privacy preferences in a certain area.

Request compliance with privacy preferences	
URL	/privacy-preference/add/
Request method	POST
Description	If the user has been in sensor range during a recording, the deletion or filtering of a recording according to his/her privacy preferences will be performed.

iBeacon Server

Start broadcasting	
URL	/beacon/broadcast/
Request method	POST
Description	Expects a JSON object containing a major and minor value as well as a UUID.

Stop broadcasting	
URL	/beacon/stop_broadcasting/
Request method	POST
Description	Stops the device from broadcasting.

Central Storage

Add a presence token	
URL	/presence-token/
Request method	POST
Description	The central storage receives presence tokens via this API.

5.6 Summary

In this chapter we described our implementation of key components and concepts. The implementation comprises an Android application and three Django web servers making heavy use of REST APIs for internal communication. The system can record a webcam's video stream and transfer it to the central storage component. Furthermore, it features the previously described access control mechanism based on presence tokens and disseminates them via a self made iBeacon device. Future developers can easily add other presence sources and sensors due to the system's modularisation. The implementation will build the foundation for the following investigation on the system's scalability and future development.

6 Evaluation

6.1 Overview

The previous chapters have described a prototype implementation and the system's design based on the requirements stated in chapter 2. In this chapter we will evaluate the developed system. The evaluation has two parts: Firstly, a quantitative evaluation showing the system's scalability by measuring the prototype's response times depending on several factors. Secondly, a qualitative analysis on the system's feasibility in terms of operational costs.

6.2 Performance Analysis

6.2.1 Overview and Testbed Configuration

In this chapter, we present our evaluation of the system's scalability (R4). The system must process large quantities of data and operate with a large number of attached devices. Therefore, processing times should increase no more than linearly with the amount of attached sensors and presence sources or stored recordings. Our focus was on the type of growth (e.g. linear, exponential) with which processing times increase rather than absolute times since the system was not running on a high-end machine or optimised for fast responses.

To measure processing times we needed to generate test data comprising sensors, presence sources and recordings. Sensors and presence sources were attached via the system's plugin interface, which caused the system to behave as if actual devices were attached. However, these sensors were not providing any data. Otherwise, our test cases would generate a vast quantity of data since they simulate several thousand sensors. This would clearly exceed the computational power and storage we had available. We assume that the missing data stream will not change the type of growth with which the processing times increase since recordings are just stored and not further processed. Again, we are focusing on the type of growth rather than absolute processing times. Two types of recordings have been used: plain text (e.g. as provided by sensors such as thermometers); and video files. However, we could only store a restricted amount of video files due to storage limitations.

All tests were performed on a virtual machine maintained by the Lancaster University's datacenter. The virtual machine ran Ubuntu 14.04 and had 2GB RAM, 23GB hard disk and

two Intel Xeon CPU's with 3.47GHz clock frequency available. All performance tests were conducted using the sensor management component. We did not run a performance analysis with the personal device since it only stores presence tokens and encryption keys using a FIFO queue and is not directly affected by an increasing number of sensors or recordings. The central storage component receives presence tokens from personal devices and retrieves recordings from the sensor management component. Due to its prototypical nature it does not perform queries on recordings that could affect the system's performance. Hence, it is not interesting for a performance evaluation.

We measured processing times depending on the amount of

- video recordings stored.
- text based recordings stored.
- sensors attached.
- broadcasters attached.

Measured processing times include: sending a request to the sensor management component's REST-API; receiving an acknowledgement; the back-end preparing the downloadable tar archive; sending requests to check if the recording is downloadable; receiving the recording's processing status and an acknowledgement that the requested recording is downloadable.

6.2.2 Measured Response Times

All measurements involved the sensor management component's REST-API (see chapter 5.5.4). In particular, we were using all API calls used to retrieve recordings: "Request a recording", "Get the processing status of a request for a recording", "Download a recording". Measurements were taken using Python scripts, which output mean and standard deviation. We took 200, 500 or 1000 samples for each data point. The number of samples depended on two factors: the time necessary to take them; the amount of noise in previous samples. We were interested in the system's behaviour while it is in operation rather than its behaviour after start or longer idle periods. Therefore, we took 50 preceding samples which were not considered by the calculations. The script has been executed on the same machine than the server under test. While this made the tests less realistic it enabled us to avoid random network impacts. As already stated above we were more interested in the type of growth with which processing times increase rather than absolute times.

The detailed evaluation results are shown in table 6.1 and 6.2 and visualised in figure 6.2 to 6.4. In all cases except for one the processing times increase linearly with the amount of devices attached or recordings stored. The standard deviation for processing times depending on the number of attached broadcasters and sensors increase steadily. We hypothesise that this behaviour is due to an increasing workload. The system has to prepare the recording of

Number of sensors/ broad-casters/ recordings	Processing time depending on the amount of attached sensors [ms]	Processing time depending on the amount attached broad-casters [ms]	Processing time depending on the amount of stored text based record-ings [ms]	Processing time depending on the amount of stored video recordings [ms]
1	58.99	65.19	62.06	2322.62
5	58.90	80.02	60.45	2357.92
25	61.84	69.08	61.49	2333.89
50	64.51	91.41	60.60	2518.13
75	75.60	74.39	61.07	2267.59
100	76.52	74.80	62.40	1986.96
200	79.79	81.69	62.07	
400	95.43	98.81	63.20	
600	109.64	119.30	62.53	
800	115.27	107.78	62.91	
1000	123.76	120.59	64.40	
1200	128.29	128.58	65.10	
1400	135.80	133.30	65.55	
1600	136.56	136.16	64.75	
1800	135.61	142.09	65.28	
2000	140.18	156.67	66.16	

Table 6.1: This table shows the performance evaluation's results. We took 500 samples per data point in the first two columns, 200 for the third and 1000 for the fourth

sensor data and broadcasting of presence token, which includes for example the creation of presence token and database entries. An increasing workload might cause variations in the system's processing times and therefore lead to an increasing standard deviation.

As shown in table 6.1 processing times for videos are much higher than for text based recordings. After receiving a request the back-end has to create a tar archive containing the requested recording and move it to the appropriate folder. Hence, this discrepancy can be explained with the much higher file size of video recordings. We note that above 50 recordings, the processing time for video recordings decreases (see figure 6.4). This unexpected result was replicated in repetition of the study. Our expectation is that on average the processing time will increase with the amount of video recordings stored, and we cannot identify a cause for the unusual behaviour observed.

In summary, the results suggest that the processing times increase linearly with the amount of devices attached and recordings stored. This growth has an upper bound for two reasons.

Number of text recordings	Processing time depending on the amount of stored text based recordings [ms]
1000	59.98
2000	61.22
3000	62.88
4000	64.69
5000	66.88
6000	69.37
7000	71.47
8000	72.80
9000	74.51
10000	76.92

Table 6.2: This table shows the performance evaluation's results. The table shows processing times depending on large amounts of recordings stored. We took 500 samples per data point.

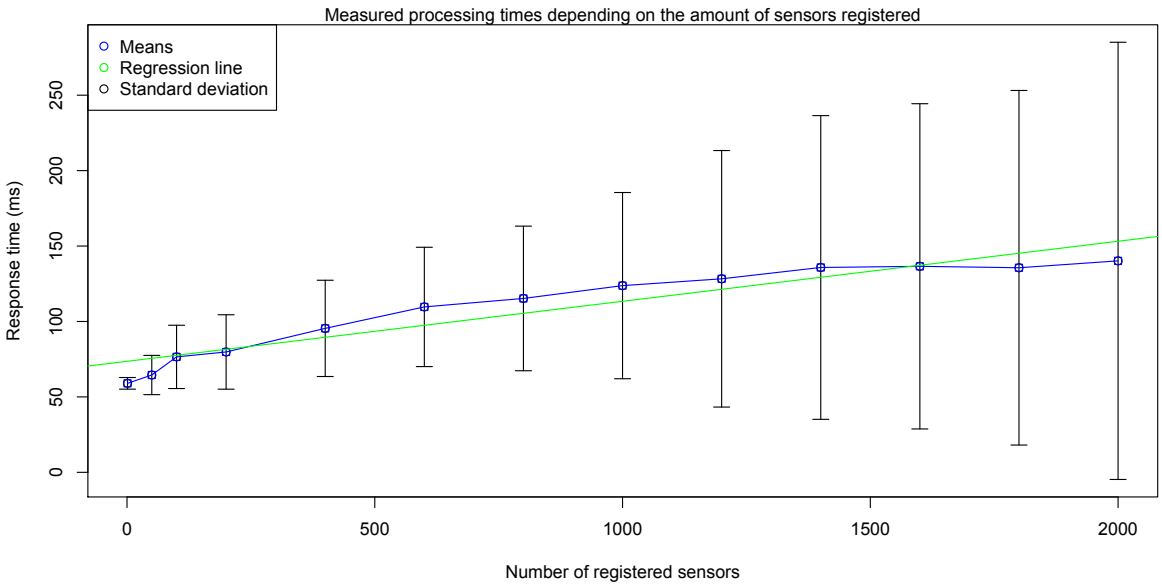

Figure 6.1: This graph shows measured processing times depending on the amount of attached sensors. We took 500 samples per data point.

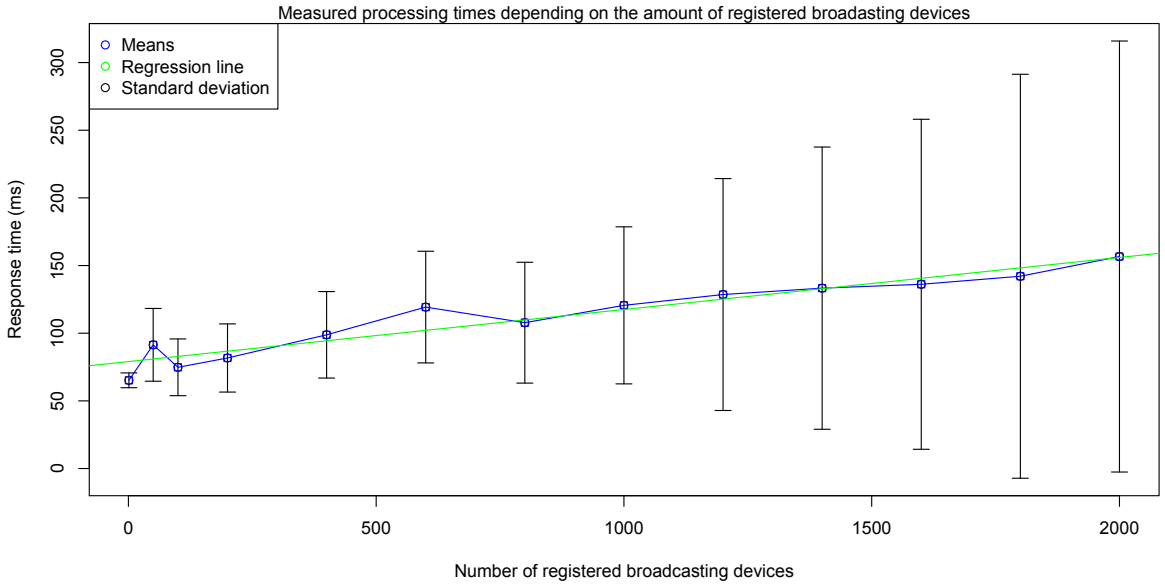

Figure 6.2: This graph shows measured processing times depending on the amount of attached broadcasters. We took 500 samples per data point.

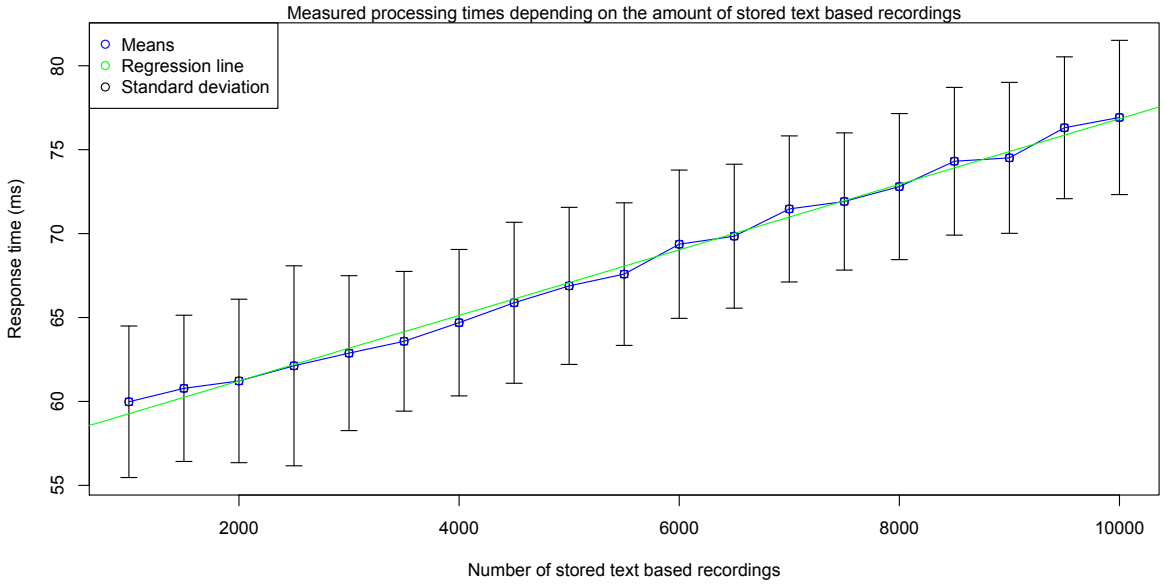

Figure 6.3: This graph shows measured processing times depending on a large number of stored text based recordings. Wee took 500 samples per data point.

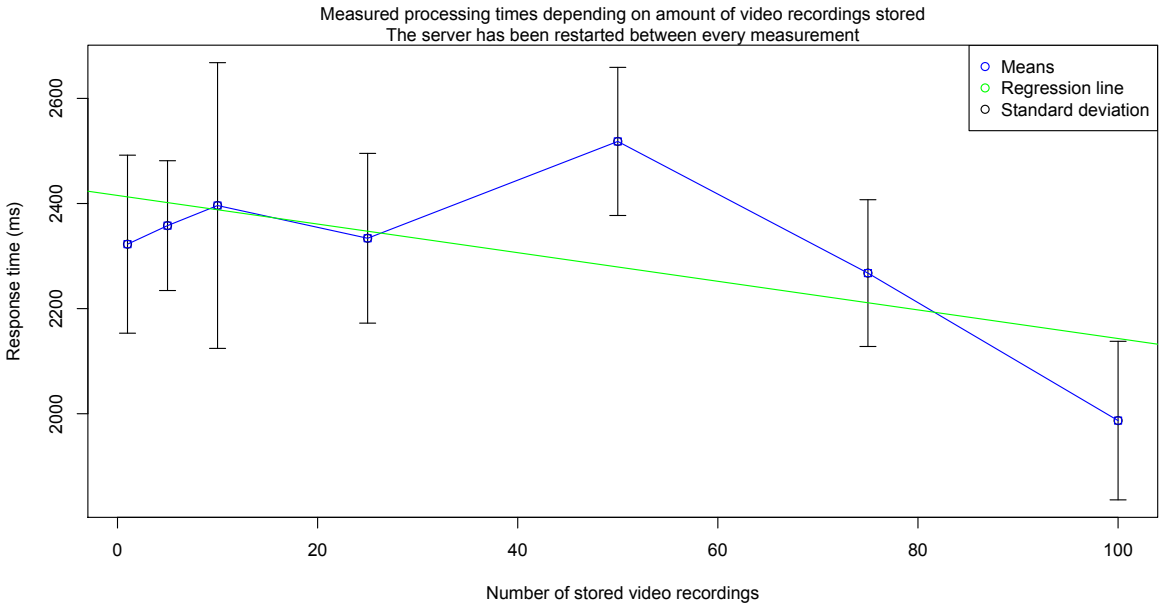

Figure 6.4: This graph shows measured processing times depending stored video recordings. We took 1000 samples per data point.

Firstly, the sensor manager component acts as cache and deletes recordings after a certain amount of time and therefore the amount of stored recordings has a maximum. Secondly, a sensor management component connects to devices in a certain area. Since this area is constrained the maximum number of devices in it has a natural limit.

The absolute response times seem reasonable, keeping in mind that each sample measures the total processing time of multiple REST API calls, which in turn causes file manipulations and several database queries. Furthermore, the implemented system is not optimised for fast responses and does not run on a high-end machine. Due to the REST-API's asynchronous design, actual response times to HTTP requests are even faster than the measured total time involving several HTTP requests. Furthermore, measured critical processing times increased at maximum linearly by the scale at which the system operates. Therefore, our results indicate that the system functions on a small as well as on a large scale.

6.3 Cost Assessment

The system designed in this thesis has high demands on bandwidth and storage capacity. This is the case since it constantly records, stores and transfers sensor data including video feeds. In this section we investigate the system's operational costs and possible business models. We

Office building		**Household**		**Restaurant**	
Number of people	360	Number of people	3	Number of people	30
Number of storeys	3	Number of rooms	4	Number of storeys	2
Offices per storey	60	Lower bound of cameras per room	1	Number of cameras per storey	4
Lower bound of cameras per office	1	Daily amount of hours an employed person is at home and awake (week-days/week end)	5/8	Number of hours open per day	6
Core hours of operation per day	6				
GB per 30 days	68040	GB per 30 days	1476	GB per 30 days	3024
GB per 30 days and person	189	GB per 30 days and person	492	GB per 30 days and person	100.8

Table 6.3: This table shows the estimated quantity of data generated by the system in three different scenarios. The first rows show assumptions made for each scenario.

first calculate data generated per month in three different scenarios. Based on those figures we then estimate the costs of operating the system on Amazon's Elastic Compute Cloud (EC2) and finally outline possible ways to fund the system.

Table 6.3 shows our estimations. The first rows show the assumptions we made for each scenario the last one shows the calculated amount of data in GB per month and person. We calculated the amount of GB per month independently from the the amount of persons since the system avoids duplicates and does not store recordings on a per person basis. Furthermore, we assume that a one hour long video recording is 2.1GB in size (based on the bit rate Youtube uses for 720p videos [Gooa]).

Table 6.4 shows costs caused by network traffic and storage consumption. As reference we used the prices for Amazon's EC2 service [Ama]. Costs per outgoing TB fall in descending price brackets with the total amount of outgoing traffic. For our calculations we used the lowest price since we assume that the system generates enough network traffic and makes further costs caused by traffic falling in the preceding price groups negligible. Finally, the costs per TB and month are 50$ for outgoing traffic and storage. The monthly storage costs will increase steadily since the system constantly adds recordings to the already existing ones.

The costs shown in table 6.4 could be covered by the user via monthly payments. However, increasing costs caused by amassed recordings make this infeasible on the long-term. Costs may be reduced by negotiating a custom contract or by running a data centre rather than

Scenario	Ingoing traffic ($/month)	Outgoing traffic ($/month)	Storage ($/month)
Office building	0	9.45	9.45
House hold	0	24.60	24.60
Restaurant	0	5.04	5.04

Table 6.4: This table shows costs for running the system via Amazon EC2. The calculated storage and bandwidth costs are the same.

using a third party one. Some sensor providers might be willing to pay for the service if they have an interest in publishing their recordings like universities and event organisers might do. Furthermore, not only single but comprehensive recordings might be of value. For instance city planners or product designers might be interested in recordings revealing the flows of people or how customers use products and when they struggle with them. A further way to fund the system could be advertisements. Companies could use the collected data to personalise advertisements or to shape a users electronic memory. For instance, positive memories associated with a company's product could be forwarded to the user with a higher priority than other memories. However, the social, ethical and psychological implications of these business models will need to be addressed by future research.

6.4 Summary

In this chapter we investigated the system's scalability and financial feasibility. We checked to which component's the requirement for scalability applies and identified the sensor management component as suitable for a performance evaluation. Subsequently, we set up a testbed using Python scripts and the Lancaster University's datacenter. Since we wanted to show the system's scalability we were primarily interested in the processing time's growth rather than absolute times. We took measurements depending on three different distinctive variables: amount of sensors attached, amount of presence sources attached, amount of recordings stored. The result's show that in the worst case, we see only linear growth thereby indicating that the system fulfils the requirement for scalability.

Considering the storage and bandwidth demands caused by the vast sensor network connected to the system, the question for its financial feasibility immediately arises. We estimated the amount of data generated in three different scenarios: office building; household; and restaurant. Using these we calculated the bandwidth and storage costs with Amazon's EC2 service. While the costs for a single month seem very low, storage costs for successive months will rise rapidly due to amassed recordings. This makes fully funding by the user unrealistic. Therefore we outline other possible ways to fund the system.

7 Related Work

In this chapter, we provide an overview of previous research in the areas of smart environments, memory augmentations and privacy protection.

7.1 Smart Environments

During the Stanford iRoom project, a prototype for a smart workspace, which includes large sized displays and portable computers, has been built [FJHW00]. Similar to the middleware built in this thesis, it aims to connect a heterogeneous set of data sinks and sources on the fly. While this thesis focuses on sensors, the iRoom project uses mainly actuators and software systems as data sources. Attempts to protect the iRoom's security are described in [FJHW00]. In contrast to this thesis, however, its focus is not on privacy protection.

An outcome of the iRoom project is the Interactive Room Operating System (iROS) described in [BRTF02]. It uses a modified tuple space to establish communication between different devices. We did not use the Tuple space paradigm due to synchronisation problems caused by longer disconnects of mobile devices. Modifications of the Tuple space paradigm tailored to mobile applications [Wad99] could not accommodate this.

HP's Cooltown project enables users to seamlessly interact with physical devices via virtual representations [PBC+01]. Beacons are functioning as links between both and can be either of virtual or physical nature using, for example, infrared, radio transmission or GPS. Similar to the system designed in this thesis, CoolTown purposefully uses a beacon's range to make only services in the user's vicinity available. However, this is only a convenience feature. Once the user has accessed a service he/she can bookmark the URL and access the service elsewhere. In contrast to that, our system implements strict access control using a similar mechanism.

The Microsoft EasyLiving project aims to enhance work and living spaces with a consistent user experience across multiple devices [SKB+98]. Making heavy use of cameras it aims to accomplish several tasks including: the transfer of user interfaces while the user is moving; child and pet care; as well as home automation. The system tracks and identifies users in order to provide convenience features like the personalisation of the environment. Suggested solutions to privacy issues arising with that include the avoidance of video streams across the network or a mode in which users are only identified on their request. However, this presumably deactivates convenience features. Our system, by design, does not track or identify

users. This is the case because it has no understanding of the recorded data and can not link requests for data streams to certain locations or users.

7.2 Memory

The Microsoft SenseCam is a neck worn camera which captures pictures periodically or based on changes in the environment [HWB+06]. During a clinical trial, an amnesia patient used the SenseCam and reviewed taken pictures periodically. The final results showed a significant improvement in her ability to recall events. We are pursuing a complementary approach to worn sensors like the SenseCam by using fixed infrastructures sensors. This can have benefits in terms of recording quality, maintenance and cost-efficiency as discussed in [CMD14]. For instance, an obvious problem with worn cameras is that they easily point in misleading directions due to the user's body posture or are occluded by scarfs and other clothing.

Schmidt et al. suggest a system which reduces stress and confusion while encountering new situations by enabling the user to build up a familiarity with them in advance [SLDW14]. For instance, prior to visiting a new place the system can display relevant information and pictures of it. The suggested system could use the middleware designed in this thesis as a data source. Firstly, for content like pictures forwarded to the user. Secondly, to learn from common behaviour and needs in order to select most relevant information as described in the paper.

The combination of different data sources including worn and environmental sensors is also described in [DFC+15]. Davies et. al. describe a pervasive memory architecture with several data sources feeding data into a storage and processing component. This in turn is connected to personal devices and the public infrastructure as data sinks. While this architecture is very similar to our's, it is not as detailed and more of a high level description.

A memory augmentation prototype has been developed during the iClips project [CJ10]. It allows the user to query recorded data via context information like people nearby, weather condition, location, date and time. Its user interface is tailored to the application domain and therefore aims to ease the browsing of vast data sets. Furthermore, search results are enhanced with memory cues enabling the user to more easily identify what they are looking for. In general, the project focuses on a prototype implementation and user interaction with recorded data as well as search algorithms for lifelogging data [KJ07].

Memories occur in third as well as first person [NN83]. The findings of Nigro et al. indicate that third person view memories occur more often if people try to remember facts. Emotional events in contrast tend to lead to first person view memories. The likelihood with which one or another occur could also depend on how far the original event is in the past. Memory augmentations therefore need to provide both views to best mimic human memory. Our system can accomplish that. It provides third person views via cameras worn by other users

and installed in the environment, as well as first person views via cameras worn by the user himself/herself.

7.3 Privacy Protection

Rather than guaranteeing privacy, the privacy awareness system (PawS) discussed in [Lan02] enables the user to protect his/her privacy in ubiquitous computing environments. Service providers collecting data about the user can ask for his/her privacy preferences and react correspondingly. Those agreements are recorded and in case of a violation used to hold the culprit accountable. Langheinrich suggests that interest groups could offer downloadable privacy preferences in order to remove the burden of creating and maintaining them from the user. This is also the case for our system. For certain situations a third party can easily choose a reasonable privacy preference. For instance, the deactivation of video recordings in restrooms. Furthermore, PawS uses a similar technique to announce services via beacons. As discussed in chapter 4.6 we complement that with a map based user interface allowing the user to look up services before he/she enters an area.

Using protocols described in [KZS02] a service can implement access control based on the user's context. The protocol exploits constraints like transmission range certain channels have. Kindberg et al. describe a basic protocol involving a client, server and a proxy, which communicates with the client via a constrained channel. On top of that, two extensions protecting the user's privacy and covering a special case in which the server can't communicate with the proxy are described as well. The access control mechanism based on presence tokens described in this thesis is heavily influenced by the basic protocol. However, we authenticate the user's previous context rather than his/her current context. Besides that, a token received via a constrained channel is not directly sent back to the broadcasting service. In order to protect a user's privacy, we introduced a third party to do that for many users and therefore mask the individual user.

A middleware protecting sensible data about users is described in [MFD03]. The system receives queries for users' data and decides whether to disclose them. This decision can depend on various factors ranging from the requesting institution, purpose and retention to whether the requester has a commercial or non profit character. The requesters privacy policies sent along with requests are than compared to the user's privacy preferences. This is done by validator components, which eventually determine if a request is accepted or declined. An inconclusive validator can require other validators possessing further sets of privacy preferences or data sources on which the decision has to be based. In our system recordings might be spread across several sensor management components from where third parties can request them. The fact that those are deliberately not linked to recorded persons in order to protect their privacy would exacerbate the decision which validator to choose. Furthermore, in contrast to

the location data tailored system described in [MFD03], our system is required to protect an arbitrary set of sensible data including video and audio recordings.

How location based services can be offered in ubiquitous computing without making the user traceable is addressed in [BS03]. To achieve that, Beresford et al. argue that updates on the user's location should be bound to changing short-term pseudonyms instead of the user's identity or long-term pseudonyms. However, if the location system's resolution is high enough, a malicious service can link changing pseudonyms and therefore deanonymize the user. Beresford et al. introduce the concepts of application and mix zones to tackle this, whereby location updates are sent in the first but not in the latter one. This aims to make users in the mix zone indistinguishable and therefore pseudonyms used before and after entering the mix zone unlinkable. Their further investigation showed that the anonymity provided by mix zones depends on their context, size and how populated they are. Our system however needs to ensure the user's privacy in arbitrary settings and can therefore not be based on mix and application zones. Furthermore, as discussed in 4.8 we require the user to collect presence tokens rather than disclosing his actual position to any part of the system.

7.4 Summary

This thesis focuses on the privacy aware collection of data from fixed infrastructure and mobile sensors for memory augmentations. This chapter described previous work in the area of smart environments, which faces similar challenges in terms of device heterogeneity and location based services as we do. Subsequently, we gave an overview on current and previous work focusing on human memory. This includes work encouraging the use of wearable and fixed infrastructure sensors in order to provide recordings from different perspectives. Finally, we discussed previous research on privacy protection in ubiquitous computing ranging from Langheinrich's PawS and a context authentication protocol developed during the HP Cool Town project to mix and application zones.

8 Conclusion

8.1 Overview

This Bachelor thesis has been conducted in cooperation with the Lancaster University in the context of the EU-project RECALL [Rec14]. The RECALL project investigates how research results in data capture and retrieval can be used to augment human memory. Besides that, the project's targets are applications for the wider population rather than only clinical ones.

In the course of this bachelor thesis, we designed a middleware, which collects data from various sources including fixed infrastructure sensors and provides them to memory augmentations. A consistent interface allows the development of memory augmentations to be more independent from implementation details of underlaying sensors. Furthermore, it implements several techniques, which strengthen and protect the privacy of users as well as bystanders.

After an intensive literature research, we conducted the development via the waterfall process. Therefore, we stated an initial set of requirements based on scenarios in order to provide a foundation for the following phases. Based on an architecture meeting those requirements, we implemented a prototype comprising three Django web servers and an Android application. Finally, we demonstrated that the prototype meets the requirement for scalability by showing that critical processing times increase at maximum linearly with the scale at which the system operates. Besides this, we investigated the system's financial feasibility by estimating critical parts of its operational costs. In parallel, we designed an Android library, which can be used in the context of this thesis as well as with Tacita. This display appropriation system has been designed at the Lancaster University and uses a concept similar to our presence tokens. Besides processing presence tokens from various sources, our library provides means to easily specify their further handling tailored to the remaining system.

The system's design, the Android library as well as the prototype will be used for further development at the Lancaster University and the RECALL project.

8.2 Future Work

The system designed in this thesis stands in stark contrast to previous work on memory centric applications, which focus on worn sensors and are mostly of proprietary nature. Therefore, this

thesis provides opportunities for future research as well as a valuable component for upcoming memory augmentations.

Firstly, already existing memory augmentations like Keepsake [Nic14] could be connected to the system. Partially founded by the RECALL project, Keepsake has been developed at the University of Stuttgart and aims to provide the foundation for a future ecosystem of memory centric applications. It therefore features a rich plugin interface, which could be used to connect it to our middleware.

Secondly, a map based user interface providing information like installed sensors and recording guidelines has been discussed in this thesis' design section. Future work could address the appropriate visualisation of this information as well as corresponding work flows, e.g. users stating their privacy preferences on daily routes.

Thirdly, the evaluation of appropriate presence sources in different contexts is a further opportunity. For instance, wireless technologies, which don't cover a confined area, might be appropriate in public spaces where a person's field of vision or ability to hear is also not restricted to a confined area. However, these technologies are certainly not appropriate in meeting rooms where confidential meetings take place.

Fourthly, future research could address a machine readable privacy preference languages. Although there is already a significant amount of works in this field it needs to be clarified how appropriate existing privacy preference languages are for our system [CLM02a] [CLM+02b] [EFH+97] [KKH+05] [MFD03].

Finally, we suggest the evaluation of business models beyond our discussion in the evaluation section.

8.3 Closing Remarks

In the last decades we have seen an increasing amount of work in the area of lifelogging and memory augmentations. Besides their restriction to wearable sensors and lack of interoperability, they are raising sever privacy issues for bystanders as well as users.

In the course of this thesis, we developed a middleware enabling memory augmentations to utilise a diverse set of worn and fixed infrastructure sensors. Besides that, it features several privacy protection mechanisms. This includes the ability to restrict access to recordings to people potentially featured by them. Thanks to an optimisation reducing bandwidth and storage requirements dramatically in certain cases, our estimates for the system's operational costs were surprisingly low. Furthermore, we could show the system's scalability by evaluating critical processing times.

The thesis' author hopes that this will contribute to the work on future memory augmentations and allows their design in a privacy preserving way.

Appendices

A Android Library

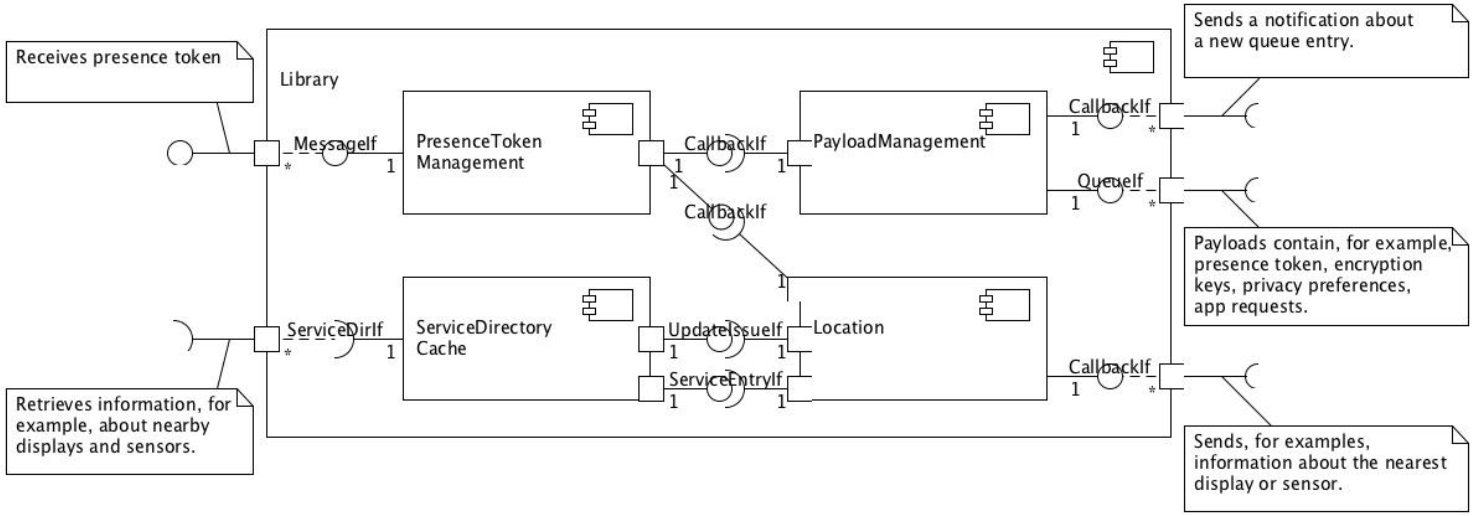

Figure A.1: This figure shows the Android library targeting the display appropriation framework Tacita and the system designed in this thesis.

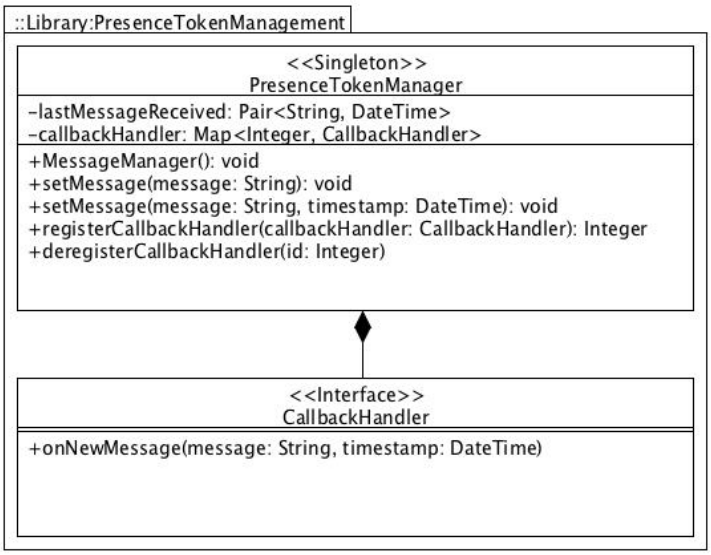

Figure A.2: This figure shows the presence token management component's classes.

Figure A.3: This figure shows the location component's classes.

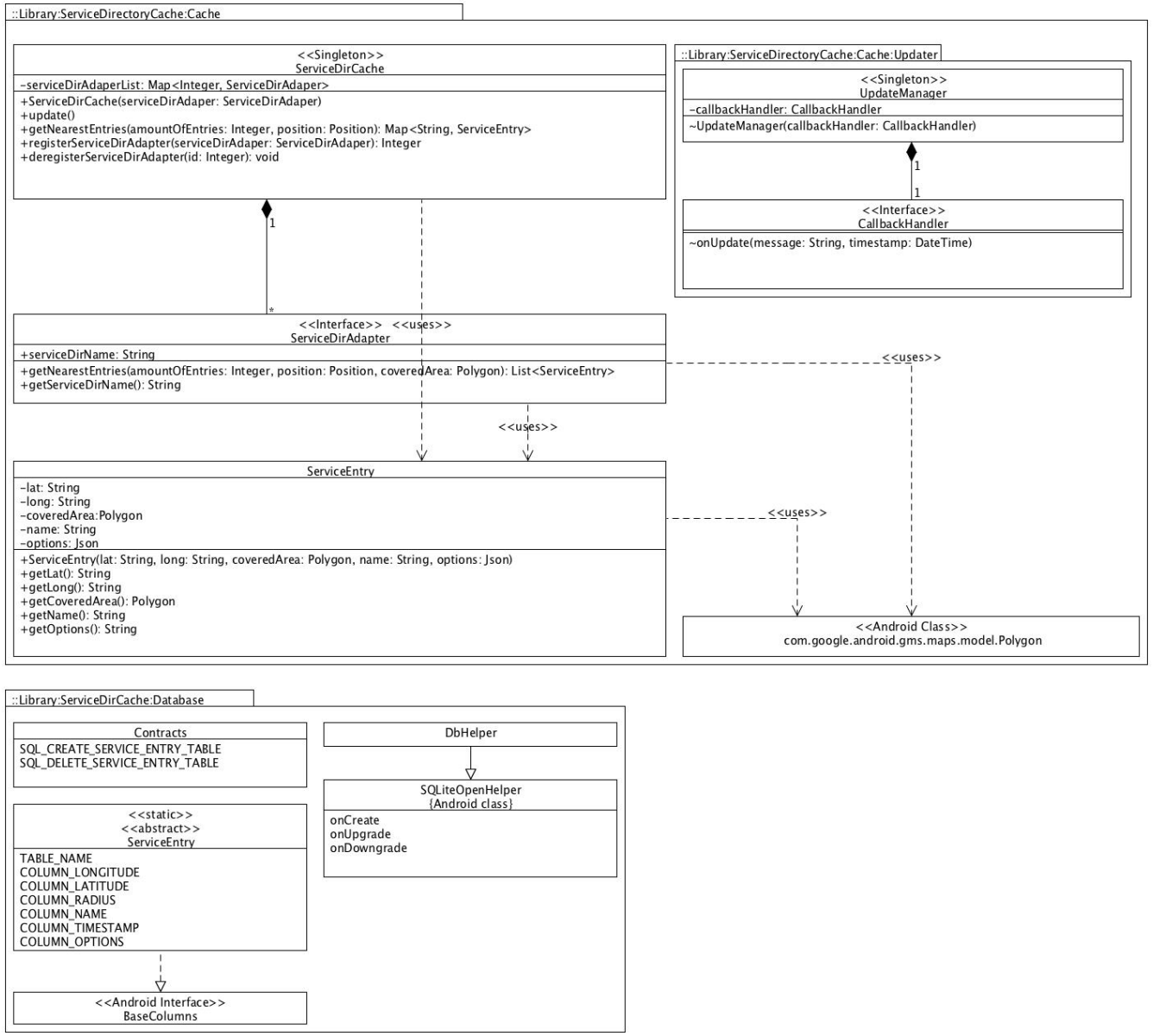

Figure A.4: This figure shows the service directory cache component's classes.

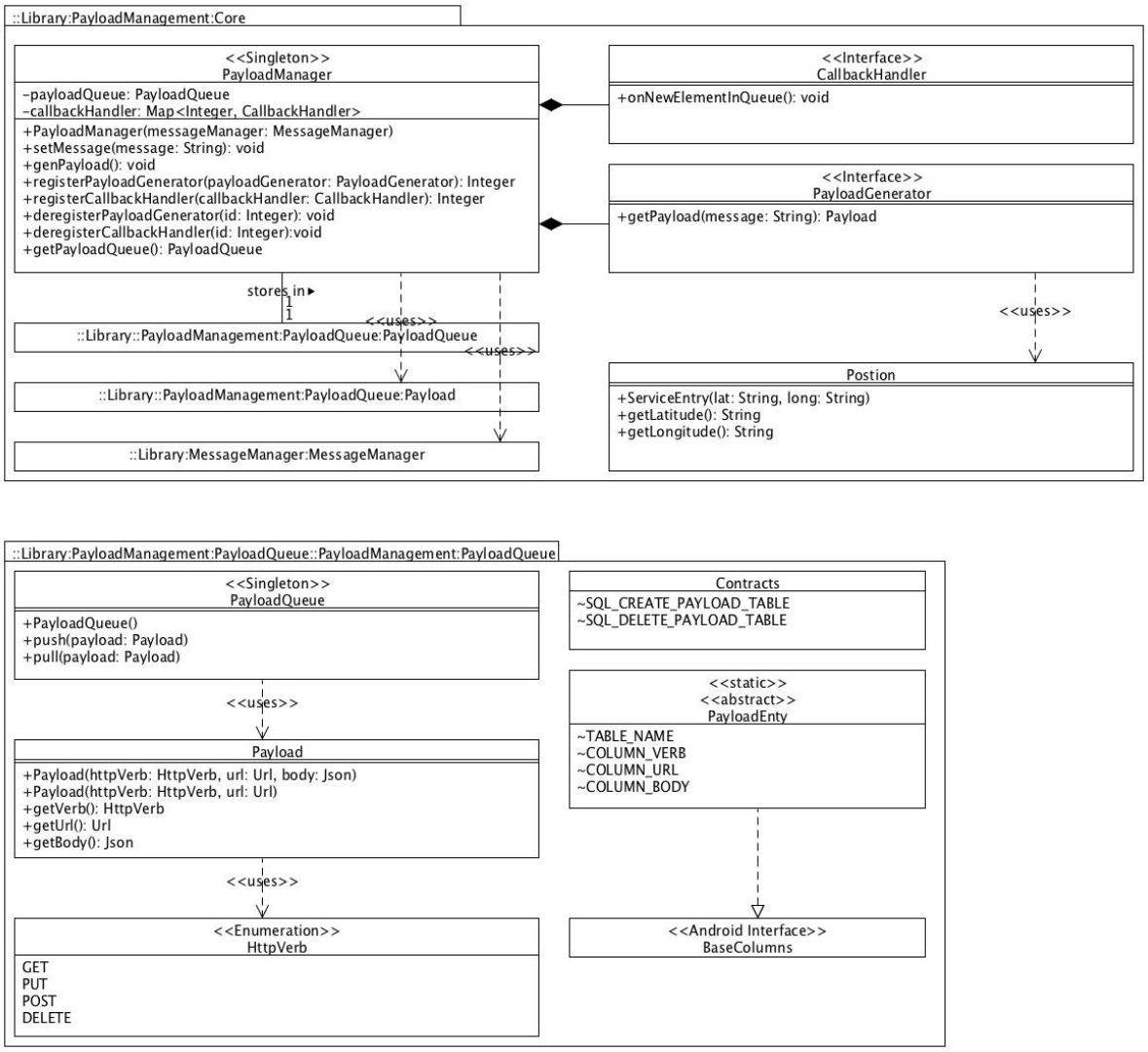

Figure A.5: This figure shows the payload management component's classes.

B Zusammenfassung

Im Laufe der letzten Jahre konnten wir eine steigende Anzahl Lifelogging Produkte und Forschungsprojekte beobachten. Diese reichen von frühen Arbeiten von Steve Mann und Microsofts MyLifeBits Projekt bis zu an Kleidung befestigten Kameras wie dem Autographer and Narrative Clip.
Diese und andere Arbeiten konzentrieren sich hauptsächlich auf vom Nutzer getragenen Sensoren und weniger auf Sensoren in dessen Umgebung. Wie vom Autor in [CMD14] diskutiert hat dies einige Nachteile, wie zum Beispiel unscharfe Bilder, verursacht durch die Bewegungen des Nutzers oder von Kleidungsstücken verdeckten Kameras und versagende Batterien.
Im Laufe dieser Bachelorarbeit haben wir einen ergänzenden Ansatz verfolgt. Wir haben eine Middleware für Gedächtnis orientierte Anwendungen entwickelt, die Daten durch ein breites Spektrum aus Sensoren bereitstellt. Der Fokus lag dabei auf Skalierbarkeit und dem Schutz der Privatsphäre von Nutzern, da das System flächendeckend Kameras und andere Sensoren einsetzt. Durch verteiltes Speichern von Sensordaten und die Vermeidung von Duplikaten konnten wir die durch das System übertragene Menge von Daten reduzieren.
Parallel dazu haben wir eine Android Bibliothek zur Verarbeitung von standortbezogenen Daten entworfen. The Android Bibliothek kann mit unserem System und dem auf Displays konzentrierten Framework Tacita das an der Lancaster University entwickelt wurde eingesetzt werden. Unsere Evaluation hat die Skalierbarkeit des Systems gezeigt, da Verarbeitungszeiten höchstens linear mit der Anzahl Eingabedaten wachsen. Dabei schlossen unsere Messungen die Simulation von bis zu 10000 Datensätzen und 2000 Sensoren auf einem Netzwerkknoten des Systems ein. Darüber hinaus haben wir die finanzielle Machbarkeit des Systems untersucht und dessen Betriebskosten geschätzt.

Bibliography

[Ama] Amazon.com Inc. AWS | Amazon EC2 | Pricing. URL http://aws.amazon.com/ec2/pricing/. (Cited on page 59)

[App] Apple Inc. iBeacon for Developers - Apple Developer. URL https://developer.apple.com/ibeacon/. (Cited on page 35)

[App14] Apple Inc. OS X Mavericks: About FileVault disk encryption, 2014. URL http://support.apple.com/kb/PH13728. (Cited on page 45)

[Atl] Atlassian. Customers. URL https://www.atlassian.com/company/customers. (Cited on page 24)

[Aut] Autographer. Home - Autographer - The World's First Wearable camera. URL http://www.autographer.com/. (Cited on page 11)

[B+45] V. Bush, et al. As we may think. *The atlantic monthly*, 176(1):101–108, 1945. (Cited on page 11)

[Blu] Bluetooth SIG Inc. Bluetooth Smart. URL http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx. (Cited on page 25)

[BRTF02] J. Borchers, M. Ringel, J. Tyler, A. Fox. Stanford interactive workspaces: a framework for physical and graphical user interface prototyping. *Wireless Communications, IEEE*, 9(6):64–69, 2002. (Cited on pages 11 and 61)

[BS03] A. R. Beresford, F. Stajano. Location privacy in pervasive computing. *Pervasive Computing, IEEE*, 2(1):46–55, 2003. (Cited on page 64)

[Cas14] S. Cass. Top 10 Programming Languages - IEEE Spectrum, 2014. URL http://spectrum.ieee.org/computing/software/top-10-programming-languages. (Cited on page 20)

[CJ10] Y. Chen, G. J. Jones. Augmenting human memory using personal lifelogs. In *Proceedings of the 1st Augmented Human International Conference*, p. 24. ACM, 2010. (Cited on page 62)

[CKDL12] S. Clinch, T. Kubitza, N. Davies, M. Langheinrich. Demo: using mobile devices to personalize pervasive displays. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 491–492. ACM, 2012. (Cited on page 12)

[CLM02a] L. Cranor, M. Langheinrich, M. Marchiori. A P3P Preference Exchange Language 1.0 (APPEL1.0), 2002. URL http://www.w3.org/TR/P3P-preferences/. (Cited on pages 31 and 66)

[CLM+02b] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, J. Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, 2002. URL http://www.w3.org/TR/P3P/. (Cited on pages 31 and 66)

[CMD14] S. Clinch, P. Metzger, N. Davies. Lifelogging for 'observer' view memories: an infrastructure approach. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pp. 1397–1404. ACM, 2014. (Cited on pages 4, 12, 62 and 75)

[Cou] G. Couturler. Welcome to Flickr - Photo Sharing. URL https://www.flickr.com/. (Cited on page 20)

[Cro] T. Crosley. isort 3.9.0 : Python Package Index. URL https://pypi.python.org/pypi/isort. (Cited on page 43)

[Dan] Daniel Greenfeld, Audrey Roy and Two Scoops Press. Django Packages : Django Packages reusable apps, sites and tools directory. URL https://www.djangopackages.com/. (Cited on page 23)

[DFC+15] N. Davies, A. Friday, S. Clinch, C. Sas, M. Langheinrich, G. Ward, A. Schmidt. Security and Privacy Implications of Pervasive Memory Augmentation. *Pervasive Computing, IEEE*, 14(1), 2015. (Cited on page 62)

[Dis11] Disqus. The Numbers of Disqus, 2011. URL http://blog.disqus.com/post/5192492910/the-numbers-of-disqus. (Cited on page 24)

[Dis13] Disqus. Scaling Django to 8 Billion Page Views, 2013. URL http://blog.disqus.com/post/62187806135/scaling-django-to-8-billion-page-views. (Cited on page 24)

[Djaa] Django REST framework - Web APIs for Django. URL http://www.django-rest-framework.org/. (Cited on page 43)

[Djab] Django Software Foundation. DjangoSuccessStoryBitbucket. URL https://code.djangoproject.com/wiki/DjangoSuccessStoryBitbucket. (Cited on page 24)

[Djac] Django Software Foundation. The Web framework for perfectionists with deadlines. URL https://www.djangoproject.com/. (Cited on pages 20 and 23)

[Dja05] Django Software Foundation. Writing and running tests, 2005. URL https://docs.djangoproject.com/en/dev/topics/testing/overview/. (Cited on page 43)

[DLC+14] N. Davies, M. Langheinrich, S. Clinch, I. Elhart, A. Friday, T. Kubitza, B. Surajbali. Personalisation and privacy in future pervasive display networks. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pp. 2357–2366. ACM, 2014. (Cited on pages 26, 30, 31 and 35)

[EFH+97] C. Evans, C. D. Feather, A. Hopmann, M. Presler-Marshall, P. Resnick. PICSRules Specification, 1997. URL http://www.w3.org/TR/REC-PICSRules/. (Cited on pages 31 and 66)

[Fac] Facebook Inc. Instagram. URL http://instagram.com. (Cited on page 20)

[Fit] Fitbit Inc. Fitbit. URL http://www.fitbit.com/uk. (Cited on page 11)

[FJHW00] A. Fox, B. Johanson, P. Hanrahan, T. Winograd. Integrating information appliances into an interactive workspace. *Computer graphics and Applications, IEEE*, 20(3):54–65, 2000. (Cited on page 61)

[GBL+02] J. Gemmell, G. Bell, R. Lueder, S. Drucker, C. Wong. MyLifeBits: fulfilling the Memex vision. In *Proceedings of the tenth ACM international conference on Multimedia*, pp. 235–238. ACM, 2002. (Cited on page 11)

[Gooa] Google. Advanced encoding settings. URL https://support.google.com/youtube/answer/1722171?hl=en-GB. (Cited on page 59)

[Goob] Google. Android Studio. URL https://developer.android.com/sdk/installing/studio.html. (Cited on page 42)

[Gre13] G. Greenwald. XKeyscore: NSA tool collects 'nearly everything a user does on the internet', 2013. URL http://www.theguardian.com/world/2013/jul/31/nsa-top-secret-program-online-data. (Cited on page 12)

[Hat] H. Hattori. autopep8 1.0.4 : Python Package Index. URL https://pypi.python.org/pypi/autopep8/. (Cited on page 43)

[HWB+06] S. Hodges, L. Williams, E. Berry, S. Izadi, J. Srinivasan, A. Butler, G. Smyth, N. Kapur, K. Wood. SenseCam: A retrospective memory aid. In *UbiComp 2006: Ubiquitous Computing*, pp. 177–193. Springer, 2006. (Cited on pages 11 and 62)

[Jet] JetBrains s.r.o. Python IDE Django IDE for Web developers : JetBrains PyCharm. URL https://www.jetbrains.com/pycharm/. (Cited on page 42)

[KJ07] L. Kelly, G. J. Jones. Venturing into the labyrinth: the information retrieval challenge of human digital memories. 2007. (Cited on page 62)

[KKH+05] V. Kolovski, Y. Katz, J. Hendler, D. Weitzner, T. Berners-Lee. Towards a policy-aware web. In *Semantic Web and Policy Workshop at the 4th International Semantic Web Conference*. 2005. (Cited on pages 31 and 66)

[KZS02] T. Kindberg, K. Zhang, N. Shankar. Context authentication using constrained channels. In *Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on*, pp. 14–21. IEEE, 2002. (Cited on pages 11, 28 and 63)

[Lan02] M. Langheinrich. A privacy awareness system for ubiquitous computing environments. In *UbiComp 2002: Ubiquitous Computing*, pp. 237–245. Springer, 2002. (Cited on pages 33 and 63)

[Lan05] M. Langheinrich. *Personal privacy in ubiquitous computing*. Ph.D. thesis, Citeseer, 2005. (Cited on page 11)

[Man97] S. Mann. Wearable computing: A first step toward personal imaging. *Computer*, 30(2):25–32, 1997. (Cited on page 11)

[Mas] Massachusetts Institute of Technology. MIT Project Oxygen: Overview. URL http://oxygen.lcs.mit.edu/Overview.html. (Cited on page 11)

[McK13] A. McKay. The restful Marketplace, 2013. URL http://blog.mozilla.org/webdev/2013/02/22/the-restful-marketplace/. (Cited on page 24)

[Mem] Memoto AB. Narrative Clip a wearable, automatic lifelogging camera. URL http://getnarrative.com/. (Cited on page 11)

[Mer] Mercurial community. Mercurial SCM. URL http://mercurial.selenic.com/. (Cited on page 42)

[MFD03] G. Myles, A. Friday, N. Davies. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing*, 2(1):56–64, 2003. (Cited on pages 31, 63, 64 and 66)

[Mic] Microsoft. MyLifeBits - Microsoft Research. URL http://research.microsoft.com/en-us/projects/mylifebits/. (Cited on page 11)

[Moz] Mozilla Developer Network and individual contributors. Python. URL https://developer.mozilla.org/en-US/docs/Python. (Cited on page 24)

[Nic14] Nicholas Rush and the Keepsake team. Keepsake, 2014. URL http://projects.hcilab.org/ahem2013/. (Cited on pages 12, 21 and 66)

[Nik] Nike Inc. Nike+ FuelBand SE. Activity Tracker & Fitness Monitor. URL http://www.nike.com/us/en_us/c/nikeplus-fuelband. (Cited on page 11)

[NN83] G. Nigro, U. Neisser. Point of view in personal memories. *Cognitive Psychology*, 15(4):467–482, 1983. (Cited on page 62)

[Ope] OpenSSL: The Open Source toolkit for SSL/TLS. URL https://www.openssl.org/. (Cited on page 45)

[Ope14] OpenCV, 2014. URL http://opencv.org/. (Cited on page 46)

[Pad13] G. Padovan. The big changes of BlueZ 5, 2013. URL http://padovan.org/blog/2013/02/the-big-changes-of-bluez-5/. (Cited on page 25)

[PBC+01] S. Pradhan, C. Brignone, J.-H. Cui, A. McReynolds, M. T. Smith. Websigns: Hyperlinking physical locations to the web. *Computer*, 34(8):42–48, 2001. (Cited on page 61)

[Rec14] Recall, 2014. URL http://recall-fet.eu/. (Cited on pages 11, 20 and 65)

[RHR+01] M. Roman, C. K. Hess, A. Ranganathan, P. Madhavarapu, B. Borthakur, P. Viswanathan, R. Cerqueira, R. H. Campbell, M. D. Mickunas. GaiaOS: An infrastructure for active spaces. 2001. (Cited on page 11)

[RWC01] G. van Rossum, B. Warsaw, N. Coghlan. PEP 8 – Style Guide for Python Code, 2001. URL http://legacy.python.org/dev/peps/pep-0008/. (Cited on page 43)

[Shi06] J. Shiell. jshiell/checkstyle-idea GitHub, 2006. URL https://github.com/jshiell/checkstyle-idea. (Cited on page 43)

[SKB+98] S. Shafer, J. Krumm, B. Brumitt, B. Meyers, M. Czerwinski, D. Robbins. The new easyliving project at microsoft research. In *Proceedings of the 1998 DARPA/NIST Smart Spaces Workshop*, pp. 127–130. 1998. (Cited on pages 11 and 61)

[SLDW14] A. Schmidt, M. Langheinrich, N. Davies, G. Ward. Déjà vu–technologies that make new situations look familiar: position paper. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pp. 1389–1396. ACM, 2014. (Cited on page 62)

[UML] UML Tool for Fast UML Diagrams. URL http://www.umlet.com/. (Cited on page 42)

[Unt13] M. Untersinger. Inside the NSA's web of surveillance, 2013. URL http://www.lemonde.fr/technologies/article/2013/10/21/inside-the-nsa-s-web-of-surveillance_3499742_651865.html. (Cited on page 12)

[Vin] Vine. URL https://vine.co/. (Cited on page 20)

[Wad99] S. P. Wade. *An investigation into the use of the tuple space paradigm in mobile computing environments*. Ph.D. thesis, Citeseer, 1999. (Cited on page 61)

[Xvi14] Xvid Solutions GmbH. Xvid, 2014. URL https://www.xvid.org/. (Cited on page 46)

[You] D. G. Young. Releases RadiusNetworks/android-ibeacon-service GitHub. URL https://github.com/RadiusNetworks/android-ibeacon-service/releases. (Cited on page 25)

All links were last followed on November 21, 2014.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature